

Skripta iz premeta

Veštačka inteligencija 1

Bojana Dimić Surla

Departman za matematiku i informatiku, Prirodno-matematički fakultet, Univerzitet u Novom Sadu

Poglavlje 1

Uvod

1.1. Šta je Veštačka inteligencija?

Postoje četiri pristupa definisanju pojma Veštačke inteligencije (VI). Tabela 1.1 prikazuje ova četiri pristupa u dve dimenzije. Gornje definicije odnose se na proces mišljenja i rasuđivanja, dok se donje odnose na ponašanje. Definicije levo vrše poređenje sa čovekom, dok definicije desno vrše poređenje sa racionalnošću odnosno idealnim učinkom (sistem je racionalan ako radi "pravu stvar"). Svaki od navedenih pristupa je bio podržavan u prošlosti, od strane različitih ljudi sa različitim metodama.

Misliti kao čovek	Misliti racionalno
Ponašati se kao čovek	Ponašati se racionalno

Tablela 1.1

Ponašati se kao čovek: Tjuringov test

Turingov test napravio je Alan Tjuring 1950. godine sa ciljem da obezbedi adekvatnu definiciju inteligencije. Kompjuter polaže test ako ispitivač (čovek) posle postavljanja određenog broja pitanja (pisanjem) ne može da prepozna koje odgovore daje čovek a koje kompjuter.

Kompjuter je trebalo da pokaže sledeće sposobnosti

- **obrada prirodnog jezika** da bi mogao da komunicira na Engleskom jeziku;
- **reprezentacija znanja** da bi skladištio ono što zna i ono što sazna;
- **automatsko rasuđivanje** da bi mogao da koristi sačuvane informacije i da dođe do zaključaka i odgovora;
- **mašinsko učenje** da bi usvojio nove okolnosti i uočio nove šablone.,

Navedene discipline predstavljaju osnovu veštačke inteligencije.

Misliti kao čovek: Kognitivno modeliranje

Da bismo mogli da kažemo da neki program razmišlja isto kao čovek, moramo na neki način odrediti kako čovek razmišlja. Kognitivna nauka je interdisciplinarna nauka koja korišćenjem kompjuterskih modela iz VI i eksperimentalnih tehnika psihologije razvija teorije o ljudskom umu.

Kognitivna nauka svoje eksperimente izvršava nad životinjama i ljudima, dok VI to radi na računaru.

Misliti racionalno: Pravila zaključivanja

Grčki filozof Aristotel je prvi pokušao da objasni termin "ispravno mišljenje". Njegovi silogizmi predstavljaju šablon za pravilno donošenje zaključka. Primer: Sokrat je čovek, svi ljudi su smrtni. Zaključak: Sokrat je smrtnan. Ova pravila razmišljanja koja upravljaju operacijama uma predstavljaju početak razvoja logike.

Logičari u 19 veku su razvili preciznu notaciju za izraze koji opisuju bilo koji objekat i relacije među objektima. Do 1965. godine razvili su se programi koji u principu mogu da reše bilo koji problem predstavljen u notaciji logike. Ako ne postoji rešenje program se vrti u beskonačnoj petlji. U početku razvoja VI logičari su mislili da na bazi logike mogu da naprave celokupne inteligentne sisteme.

U ovakvom pristupu postoje dva problema. Prvi je taj da nije lako predstaviti neformalno znanje o svetu u formalnoj notaciji logike, naročito ako to znanje nije 100 posto sigurno. Drugi problem je što postoji velika razlika između rešavanja problema u principu i rešavanja problema u praksi.

Ponašati se racionalno: Racionalni agent

Agent je nešto što se ponaša na određeni način. Svaki kompjuterski program radi nešto, međutim od agenata se očekuje da radi više: da rade autonomno, da uzimaju informacije iz okruženja, da postoje u nekom dužem vremenskom intervalu, da se prilagode promeni, da kreiraju ciljeve i da teže njihovom dostizanju. Racionalni agent je agent koji se ponaša tako da teži da dostigne najbolji mogući rezultat.

U predhodnom pristupu "pravila zaključivanja" akcenat je na pravilnom donošenju zaključka. Pravilno zaključivanje ima veze sa racionalnošću, međutim racionalno ponašanje se ne sastoji samo od pravilnog zaključivanja. U nekim situacijama ne postoji ispravna stvar koju treba uraditi, ali ipak nešto treba uraditi. Takođe, postoje situacije u kojima racionalno ponašanje ne podrazumeva zaključivanje (na primer refleksna akcija povlačenja ruke sa vruće ringle).

Racionalni agenti imaju dve osnovne prednosti u odnosu na ostale pristupe. Prvo, mnogo su generalniji od pravila zaključivanja jer je pravilno zaključivanje samo jedan od mogućih mehanizama sa postizanje racionalnog ponašanja. Druga prednost je što je ovaj pristup pogodniji za naučni razvoj od pristupa misliti kao čovek i ponašati se kao čovek. Racionalnost je matematički dobro definisana i generalna.

1.2 Veštačka inteligencija i ostale nauke

Filozofija

- Da li se formalna pravila mogu koristiti za donošenje validnih zaključaka?

- Kako je iz fizičkog mozga nastao um?
- Odakle dolazi znanje?
- Kako znanje dovodi do akcije?

Aristotel je prvi formulisao precizan skup pravila koji opisuju racionalno rasuđivanje ljudskog uma. On je napravio neformalni sistem silogizama za racionalno rasuđivanje, koji omogućava generisanje zaključaka na osnovu datih premisa (na primer: premise: Sokrat je čovek. Svi ljudi su smrtni. zaključak: Sokrat je smrtni.)

Matematika

- Koja su formalna pravila za donošenje validnih zaključaka?
- Šta se može izračunati?

algoritam, teorija nekompletnosti, Tjuring: koje funkcije su izračunljive, Tjuringova mašina može da izračuna sve izračunljive funkcije

- Kako da tumačimo nepouz dane (neizvesne) informacije?

teorija verovatnoće

Razvoju veštačke inteligencije prethodili su: razvoj formalne logike (Leibniz), postavljanje osnova teorije grafova (Euler), matematička formalizacija zakona logike (Bool) i razvoj predikatskog računa prvog reda (Frege).

Ekonomija

- Kako odlučivati da bi se dobio najveći profit?

teorija odlučivanja

- Kako postupati u interakciji sa drugima?

igre (postojanje protivnika)

- Kako postupati sada kada se profit očekuje tek u budućnosti?

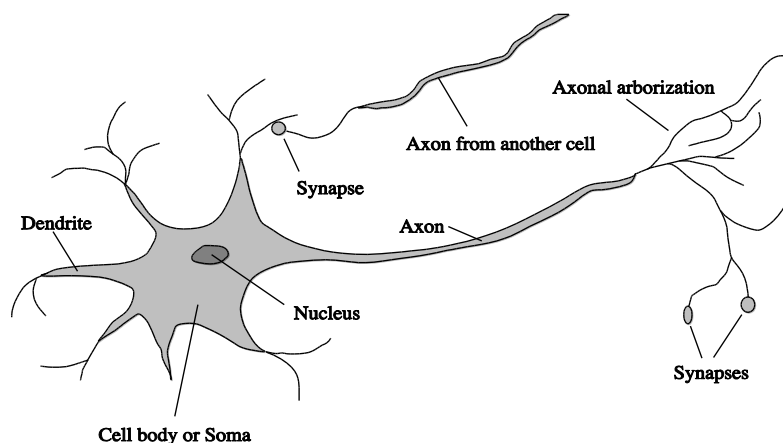
operaciona istraživanja

Neuronauka

- Kako mozak obrađuje informacije?

Neuronauka je nauka o nervnom sistemu koja se naročito bavi i mozgom. Mozak se sastoji od nervnih ćelija, neurona. Interesantno je kako kolekcija tih jednostavnih ćelija dovodi do misli, akcija i svesnosti. Mozak i digitalni računari imaju različite karakteristike. Računari imaju vremenski ciklus koji je milion

puta brži od mozga. Dok, sa druge strane, mozak ima više memorije i bolju povezenost nego najmoderniji računar. Čak i sa računarom za virtuelno neograničenim kapacitetom, ne bismo znali kako da dostignemo ljudsku inteligenciju.



Lingvistika

Računarska lingvistika i obrada prirodnog jezika predstavljaju mešavinu VI i moderne lingvistike. Ispostavilo se da je razumevanje prirodnog jezika komplikovanije nego što se u početku mislilo, jer zahteva razumevanje teme i konteksta a ne samo razumevanje strukture rečenice. Takođe, početna istraživanja o reprezentaciji znanja bila su povezana sa lingvistikom.

1.3 Istorija veštačke inteligencije

Gestacija VI (1943-1955)

Prva istraživanja koja se danas prepoznaju kao VI sprovedi su Warren McCulloch i Walter Pitts (1943). Ova istraživanja zasnivala su se na tri oblasti: osnove psihologije i funkcije neurona u mozgu, formalna analiza iskazne logike i Tjuringova teorija algoritama. Predložen je model vaštačkog neurona u kome se svaki neuron karakteriše kao uključen i isključen ("on" i "off"). Neuron se uključuje stimulacijom dovoljnog broja susednih neurona. Autori su pokazali da se bilo koja izračunljiva funkcija može izvršiti pomoću neke mreže povezanih neurona i da se svi logički operatori (and, or, not) mogu implementirati jednostavnim mrežnim strukturama. McCulloch i Pitts su tvrdili da pravilno definisana mreža može da uči.

Donald Hebb je 1949 demonstrirao jednostavno pravilo koje modifikuje jačinu veze između neurona. Njegovo pravilo, koje se danas naziva Hebovo učenje je veoma uticajno i danas i glasi:

Kada je akson neurona A dovoljno blizu neurona B, tako da ga može stimulisati, i ako se to ponavlja dovoljno često, dešavaju se takve promene i metabolički procesi u obe ćelije da je efikasnost uticaja neurona A na neuron B povećana.

Dva studenta sa Harvarda, Marvin Minsky i Dean Edmonds napravili su prvu neuralnu mrežu - SNARC 1950. godine. Ova mreža je koristila 3000 vakuumskih cevi i deo pilotskog mehanizma iz B-24

bombardera da stimuliše mrežu od 40 neurona. Mreža je rešavala problem pronalaska puta u lavirintu. Minsky je napisao doktorsku disertaciju o neuronskim mrežama, ali je komisija bila skeptična u vezi toga da li se ovakvo istraživanje može smatrati značajnim. Međutim, Neumann je na kraju konstatovao da ako sada to nije jednom će sigurno biti. Minski je kasnije dokazao veoma uticajnu teoremu čime je pokazao neke nedostatke istraživanja u oblasti neuronskih mreža.

Postojali su brojni primeri istraživanja koji se mogu okarakterisati kao VI, ali vizija Alana Turing je bila najuticajnija. On je održao predavanje 1947. godine u Londonskom Matematičkom Društvu na kom je objasnio Turingov test, mašinsko učenje, genetski algoritam i pojačano učenje. On je takođe predložio ideju dete-program koja kaže: Umesto da pokušavamo da napravimo program koji simulira um odraslog čoveka, zašto ne bismo pokušali da napravimo program koji simulira um deteta.

Rođenje VI (1956)

Princeton je bio dom još jedne uticajne figure u VI, to je John McCarthy. Nakon što je doktorirao 1951. godine i radio dve godine kao predavač, McCarthy se preselio u Stanford, pa zatim u Dartmouth College koji se smatra zvaničnim mestom rođenja VI. McCarthy je dao predlog da se okupe američki istraživači iz oblasti teorije automata, neuralnih mreža, i nauke o inteligenciji. Organizovana je dvomesečna radionica (workshop) u Dartmouth-u u leto 1956. godine u kojoj je učestvovalo 10 ljudi (sa Prinstona, iz MIT-a i IBM-a). Tema na toj radionici bila je da se nastavi sa istraživanjem na osnovu pretpostavki da se svaki aspekt učenja ili bilo koje drugi oblik inteligentne aktivnosti može tako precizno opisati da se može napraviti mašina koja ga simulira. Plan je bio da se pokuša sa pravljenjem mašina koje mogu da koriste jezik, formiraju apstrakciju i koncepte, rešavaju probleme koji se rezervisani za ljudski um i da se unapređuju.

Iako su svi naučnici na skupu imali ideje ili čak programe za specijalne namene kao što je na primer igra dame, dva naučnika sa tog skupa, Newell i Simon su već imali program za rasuđivanje, Logički Teoretičar (The Logic Theorist). Ubrzo nakon radionice ovaj program je mogao da dokaže sve teoreme iz drugog poglavlja Principa Matematika (Russell and Whitehead: Principia Mathematica). Russell je bio posebno ushićen kada mu je Simon pokazao da program izvede dokaz jedne teoreme kraće nego u Principima.

Ova radionica nije dovela do nekih bitnih pronalazaka, ali jeste spojila bitne ljude. U narednih 20 godine ovom oblasti će dominirati ovi ljudi i njihovi studenti na MIT, CMU (Central Michigan University), Stanfordu i IBM-u.

Na radionici u Dartmouth-u je postalo očigladno da VI mora postati posebna oblast i ne može se svrstati pod postojeće nauke (operaciona istraživanja, teorija odlučivanja ili matematika). Prvi razlog za to je što, za razliku od drugih oblasti, VI od početka propagira ideju o dupliciranju ljudskih sposobnosti kao što su kreativnost, usavršavanje i upotreba jezika. Drugi razlog je metodologija. VI je jedina oblast koja je jasno grana računarskih nauka i jedina oblast koja pokušava da napravi mašine koje će funkcionisati autonomno u kompleksnom, promenljivom okruženju.

Rani entuzijizam, velika očekivanja (1952-1969)

Uzimajući u obzir primitivne kompjutere i programerske alate, počaci VI bili su puni uspeha. Do tada se verovalo da računari mogu da izvršavaju samo jednostavne aritmetičke operacije i ništa više i bilo je zadivljujuće kada računar uspe da izvrši bilo šta više od toga. Intelektualna elita u to vreme je verovala da računar nikada neće moći da uradi X, a istraživači iz oblasti VI su reagovali tako što su demonstrirali jedno po jedno X.

Rani uspeh Newell-a i Simon-a praćen je izumom poznatim kao General Problem Solver (GPS). Za razliku od logičkog teoretičara, ovaj program je bio od početka dizajniran da imitira postupak rešavanja problema koji je svojstven čoveku. Na ograničenom skupu problema koji je mogao da reši, bilo je jasno da je redosled ispitivanja podciljeva i mogućih akcija GPS-a sličan ljudskom rasuđivanju. Samim tim, GPS je bio prvi program koji je realizovao pristup "misliti kao čovek". Uspeh GPS-a naveo je Newella i Simona da formulišu čuvenu pretpostavku sistema fizičkih simbola koja kaže da sistem fizičkih simbola poseduje potrebna i dovoljna sredstva za uopštene inteligentne akcije. Značenje ove pretpostavke je da bilo koji sistem (čovek ili mašina) koji ispoljava inteligenciju mora da manipuliše sa strukturama podataka sastavljenih od simbola.

Wikipedia:

Sistem fizičkih simbola (takođe poznat kao formalni sistem) uzima fizičke paterne (simbole), kombinuje ih u strukture (izraze) i manipuliše njima (koristeći procese) kako bi proizveo nove izraze.

Tvrdnja Newella i Simona nagoveštava da je ljudsko razmišljanje neka vrsta manipulacije simbolima (jer je simbolički sistem potreban za inteligenciju) i da mašine mogu biti inteligentne (jer je simbolički sistem dovoljan za inteligenciju).

Herbert Gelernter je 1959. godine konstruisao dokazivača geometrijskih teorema koji je mogao da dokaže teoreme koje su mnogim studentima matematike bile teške.

Počevši od 1952, Arthur Samuel je napisao seriju programa za igru Dame koji je na kraju naučio da igra na jakom amaterskom nivou. U toku rada na ovom programu, on je opovrgao ideju da računar može da radi samo ono što mu se kaže, jer je njegov program ubrzo naučio da igra bolje od njegovog kreatora. Program je demonstriran na televiziji u februaru 1956. godine i ostavio je snažan utisak.

John McCarthy se 1958. godine preselio iz Dartmouth u MIT i tu napravio tri velika dostignuća u istoriji VI. Prva je bila definisanje jezika Lisp koji je bio dominantan u oblasti VI u narednih 30 godina. Sa Lispom je McCarthy imao oruđe koje mu je bilo neophodno, međutim još uvek je kao problem ostao nedovoljan pristup oskudnim i skupim računarskim resursima. Kao odgovor na ovaj problem, on je zajedno sa saradnicima izumeo time sharing. Treće dostignuće je publikovanje naučnog rada pod naslovom *Programi sa zdravim razumom* u kome je opisan Advice Taker (davalac saveta) – hipotetski program koji se može smatrati kao prvi kompletan sistem veštačke inteligencije. Isto kao Logic Theorist i Geometry Theorem Prover, McCarthyjev program je bio dizajniran tako da koristi znanje da bi tražio rešenje za

problem. Međutim, on se razlikovao od prethodnih po tome što je u njega bilo ugrađeno opšte znanje o svetu.

Iste 1958. godine Minski je prešao na MIT. Između njega i McCarthyjevog pristupa postojala je velika razlika. Naime, McCarthy je zastupao ideju o predstavljanju znanja i rezonovanju zasnovanim na formalnoj logici, dok je Minski bio zainteresovan na pravljenje programa koji rade da bi na kraju razvio anti-logic pogled na VI.

Minski je bio mentor nekolicini studenata koji su radili na određenom skupu problema za čije rešavanje je bila neophodna VI. Ovaj skup problema bio je poznat pod nazivom mikrosvetovi i oni su predstavljali probne poligone za rešavanje problema VI. Najpoznatiji među njima bio je svet blokova (blocks world). Svet blokova se sastoji od skupa čvrstih blokova (geometrijskih figura u tri dimenzije) smeštenih na nekoj površini. Tipičan zadatak u ovom svetu sastoji se od premeštanja blokova na određeni način koristeći ruku robota koja može da uzme jedan blok u jednom trenutku.

U ovom periodu procvetao je i raniji rad McCulloch-a i Pitts-a na neuronskim mrežama. Pokazano je da veliki broj elemenata može kolektivno predstavljati individualni koncept.

Doza realnosti (1966-1973)

Od samog nastanka VI, istraživači nisu bili skromni u predviđanjima o dolazećem uspehu. Simon je predvideo da će u roku od 10 godina računar biti šampion u šahu i da će važne matematičke teoreme dokazivati računar. Ova predviđanja su se obistinila, ali ne za 10, nego za 40 godina.

Prvi problemi u razvoju VI pojavili su se zato što programi koji su pravljeni nisu znali ništa o problemu koji su rešavali. Oni su dolazili do rešenja pukom sintaksnom manipulacijom. Jedna od najpoznatijih priča iz ovog razdoblja opisuje rane mašine za prevođenje koje su bile velikodušno finansirane od strane Američkog Nacionalnog Istraživačkog Saveta. Cilj je bio da se ubrza postupak prevođenja ruskih naučnih radova u osvit lansiranja Sputnika 1957. godine. U to vreme se smatralo da se prevođenje može realizovati jednostavnim sintaksnim transformacijama zasnovanim na gramatici Engleskog i Ruskog jezika i zamenom reči iz elektronskog rečnika. Međutim, činjenica je da pravilno prevođenje zahteva predhodno znanje da bi se rešile nejasnoće i da bi se rečenici dao smisao. Čuveni prevod prevoda rečenice „the spirit is willing but the flesh is weak“ (duh je spreman, ali je telo slabo) bio je „the vodka is good but the meat is rotten“ (vodka je dobra ali je meso trulo). 1966. godine ustanovljeno je da ne postoji mašina za prevođenje uopštenog naučnog teksta i da je nije moguće napraviti u bliskoj budućnosti. Sva finansiranja ovog projekta su prekinuta. Danas su mašine za prevođenje nesavršene, ali se u velikoj meri koriste za tehnička, komercijalna, vladina i Internet dokumenta.

Drugi problem koji se javio u razvoju VI je u porastu obimnosti problema koje VI pokušava da reši. Većina ranih programa je rešavala problem jednostavnim isprobavanjem različitih kombinacija koraka dok se ne pronađe rešenje. Ova strategije je u početku imala dobre rezultate jer se radilo o problemima manjeg obima (na primer: mikrosvetovi su imali ograničen broj objekata i mogućih poteza, pa su i rešenja bila jednostavna). Smatralo se da rešavanje složenijih problema zavisi od razvoja hardvera (bržeg procesora i veće memorije).

Još jedna oblast VI koja se pojavila u ovo vreme je mašinska evolucija (danas poznata kao genetski algoritmi). Ova oblast je propagirala verovanje da se odgovarajućom serijom malih mutacija nad programom u mašinskom kodu može dobiti program odličnih performansi. Uprkos hiljadama sati potrošenog procesorskog vremena, skoro nikakav napredak nije postignut. Savremeni genetski algoritmi koriste bolju reprezentaciju i postižu veći uspeh.

Treći problem se pojavio zbog nekih fundamentalnih nedostataka osnovnih struktura koje su se koristile u generisanju inteligentnog ponašanja. Na primer, knjiga koju su napisali Minski i Papert je pokazala da se perceptronima (jednostavan oblik neurosnke mreže) može pokazati kako da nauče bilo šta što mogu da predstave, ispostavilo se da mogu da predstave malo toga.

Sistemi zasnovani na znanju (1969-1979)

Rešavanje problema u prvoj dekadi pojave VI sastojao se od uopštenih mehanizama za pretraživanje koji su pokušavali da pronađu kompletna rešenja u opštim problemima. Ovakav pristup naziva se slab metod. Suprotno ovom metodu je korišćenje moćnijog, domenskog znanja koji omogućava dublje resuđivanje i lakšu obradu problema koji se javljaju u uskoj ekspertskoj oblasti.

Rani primer ovog pristupa je program DENDRAL koji je pomagao organskim hemičarima da identifikuju nepoznate organske molekule analiziranjem spektrometrije mase i korišćenjem znanja iz hemije. Važnost DENDRAL-a sastoji se u tome što je to prvi uspešan sistem zasnovan na znanju (ekspertiza je dobijena korišćenjem velikog broja specijalizovanih pravila).

Ovakva vrsta sistema nazvana je ekspertski sistem. Sledeći program koji je napravljen bio je MYCIN koji je služio za uspostavljanje dijagnoze infekcije krvi. Sa oko 450 pravila MYCIN je imao bolje rezultate od nekih lekara. Ovaj program uveo je i pojam faktora sigurnosti.

Važnost domenskog znanja je naročito vidljiva kod razumevanja prirodnog jezika. Postojalo je stanovište da robusno razumevanje jezika zahteva znanje o svetu i uopšteni metod za korišćenje tog znanja.

VI postaje industrija (1980-danas)

Prvi komercijalni ekspertski sistem napravljen je 1982. godine i zvao se R1. On se koristio za konfigurisanje narudžbi za nove računarske sistema. Procenjeno je da je ovaj program do 1986. godine uštedeo kompaniji 40 miliona dolara. Do 1988. godine razvijeno je 40 ekspertskih sistema u istoj korporaciji, dok je jedna druga korporacija imala 100 ekspertskih sistema u upotrebi i još 400 u razvoju. Procenjuje se da su ovi sistemi ostvarivali uštedu od 10 miliona dolara godišnje.

1981. godine Japanci su objavili projekat pod nazivom „Peta Generacija“, desetogodišnji plan za razvoj inteligentnih sistema koji koriste Prolog. Kao odgovor na to Amerikanci su osnovali istraživački konzorcijum da bi očuvali nacionalnu konkurentnost. Velika Britanija je takođe povećala finansiranje istraživanja, ali sada iz novoosnoivane oblasti koja je nazvana Inteligentni sistemi zasnovani na znanju. U sve tri zemlje projekti nisu dostigli svoje preambiciozne ciljeve.

Usvajanje naučnih metoda u VI

U poslednje vreme događa se revolucija u pristupu istraživanju na temu VI. Danas je uobičajeno da se nova istraživanja zasnivaju na postojećim teorijama, umesto da se prave potpuno nove teorije, danas se tvrdnje zasnivaju na čvrstim teoremama i ozbiljnim eksperimentima umesto na intuiciji i više se pažnje posvećuje realnim aplikacijama nego igrama.

U oblasti prepoznavanja govora, pojavili su se pristupi koji se zasnivaju na matematičkoj teoriji, tačnije na Skrivenom Markovom Modelu (Hidden Markov model - HMM). Ovde treba napomenuti da ljudi sigurno ne koriste ovaj model za prepoznavanje govora. Model daje matematički okvir za razumevanje problema i podršku inženjerima za tvrdnje koje rade u praksi. Mašinsko prevođenje ide u istom pravcu kao i prepoznavanje govora.

Razvoj neuralnih mreža takođe prati ovaj trend. Kao rezultat novih dostignuća u razvoju neuralnih mreža nastala je grana VI koja se naziva rudarenje podataka (data mining).

U VI istraživanja počinje da se uvodi i verovatnoća i teorija odlučivanja. Formalizam Bajesove mreže je uveden da bi se omogućila reprezentacija i zaključivanje u prisustvu neizvesnosti.

Slična revolucija u pristupu VI pojavila se u robotici, računarskom predviđanju i reprezentaciji znanja.

Novija istorija VI

Ohrabreni uspehom rešavanja potproblema u VI, istraživači su počeli da razmišljaju o "kompletnom agentu". Jedno od najpoznatijih okruženja za inteligentne agente je Internet. Sistemi koji su bazirani na VI su veoma prisutni u Web baziranim aplikacijama. Takođe, koncepti VI su u osnovi mnogih Internet alata kao što su pretraživači i sistemi za preporuke.

Uprkos uspesima VI koja rešava konkretne probleme, neki od osnivača VI kao što su McCarthy i Minsky smatraju da VI treba manje da se bavi specifičnim zadacima kao što su vožnja automobila, igranje šaha, ili prepoznavanje govora, već da umesto toga treba da se vrati svojim korenima, odnosno da pokuša da napravi mašinu koja misli, uči i kreira (Simon). Ovakva težnja naziva se Ljudski nivo VI (human-level AI) i prvi simpozijum na ovu temu održan je 2004. godine.

Slična ideja nosi naziv Uopštena veštačka inteligencija koja je ustanovljena na jednoj konferenciji 2008. godine. Ova oblast VI pokušava da pronađe univerzalne algoritme za učenje i ponašanje u nekom okruženju.

Tokom 60-ih godina akcenat u istraživanjima je bio na algoritmima. Međutim neka novija istraživanja u VI govore da u nekim problemima veću pažnju treba posvetiti podacima, a manje se baviti algoritmima. Ovo je posledica postojanja velikih skupova podataka.

VI danas

Šta VI može da uradi danas? Postoji nekoliko oblasti u kojima se ona u velikoj meri primenjuje.

Robotska vozila: robotski auto bez šofera po imenu STENLEY je vozilo opremljeno kamarama, radarima, laserskim daljinometrima da bi preuzimao podatke o okruženju i u njega je ugrađen softver koji upravlja volanom, kočnicama i brzinom.

Prepoznavanje govora: putnik koji pozove Američku avionsku kompaniju da rezerviše let može imati celokupnu konverzaciju sa automatizovanim sistemom za prepoznavanje govora i vođenje dialoga.

Igre: IBM-ov DEEP BLUE program je postao prvi računarski program koji je pobedio svetskog šampiona u šahu (Garija Kasparova).

Borba protiv spamova: algoritmi za učenje svakodnevno klasifikuju preko milijardi poruka kao spamove. Proizvođači spamova stalno unapređuju svoju taktiku, teško je protiv njih se boriti statičkim algoritmom, već to mora biti onaj koji uči i stalno se unapređuje.

Logističko planiranje: Američka vojska je koristila program DART - Dynamic Analysis and Replanning Tool da bi automatizovala logističko planiranje i raspored transporta. Transportovano je oko 50000 vozila, teret i ljudi uzimajući u obzir početne tačke, destinaciju, putanje i rat na terenu. Sistem je za nekoliko sati izgenerisao plan za koje bi trebale nedelje sa stranim metodama.

Robotika: Kompanija iRobot je prodala preko dva miliona robota-usisivača (Roomba) za kućnu upotrebu. Ova kompanija je napravila i robota koji se koristio u Iraku za transport opasnog hemijskog materijala i eksploziva.

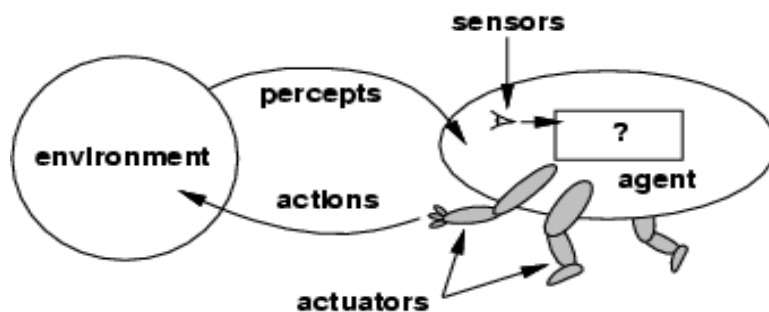
Mašinsko prevođenje: postoji program koji automatski prevodi sa Arapskog na Engleski. Program koristi statistički model koji je napravljen na osnovu primera prevoda sa Arapskog na Engleski i primera teksta na Engleskom koji ima ukupno 3 trilion reči. Niko od naučnika koji su radili na ovom programu ne zna arapski.

Poglavlje 2

Inteligentni agenti

2.1 Agenti i okruženje

Agent se može definisati kao neko ili nešto što prima informacije iz nekog okruženja preko svojih senzora i vrši neke akcije (slika 2.1). Ljudski agent ima oči, uši i druge organe kao senzore, a ruke, noge, glas za izvršavanje akcija. Softverski agent prima informacije preko tastature, čita iz fajla ili dobija informacije preko mreže a njegove akcije su prikaz podataka na ekranu, pisanje u fajl ili slanje podataka preko mreže. Robotski agent ima kamere i infracrveni daljinometar kao senzore a akcije vrši preko svojih motora.



Slika 2.1 Agent i okruženje

Treba razlikovati pojmove agentske funkcije i agentskog programa. Agentska funkcija mapira bilo koji niz zapažanja na akciju. Ona se može opisati i preko tabele, ali bi za većinu realnih agenata ta tabela bila beskonačna. Takva tabela predstavlja eksternu karakterizaciju agenta. Interno, agentska funkcija za veštačkog agenta implementira se kao agentski program. Dakle, agentska funkcija je apstraktni matematički opis, a agentski program je konkretna implementacija, izvršavanje na nekom fizičkom sistemu.

2.2 Racionalni agenti

Racionalni agent je onaj koji radi ispravnu stvar odnosno tabela agentske funkcije je popunjena korektno. Međutim, postavlja se pitanje šta znači raditi pravu stvar. Jedna od mogućih definicija odnosi se na posmatranje posledica ponašanja agenta. Agent vrši neki niz akcija u nekom okruženju, na osnovu informacija koje dobija preko senzora i kao posledica toga okruženje prolazi kroz niz stanja. Ako je taj niz stanja poželjan (odgovarajući) može se reći da agent radi pravu stvar. Stepenn poželjnosti izražava se vrednošću koja se zove mera učinka (performance measure). Treba primetiti da se racionalnost agenta meri prema stanjima okruženja, a ne prema stanjima agenta. Kako bi izgledalo kada ni agent sam odlučivao da li su njegovi postupci racionalni? Mera učinka se različito definiše u različitim sistemima.

Definicija racionalnog agenta:

Za svaki niz zapažanja, racionalni agent treba da se odluči za akciju za koju se očekuje da maksimizira njegov učinak, uzimajući u obzir niz zapažanja i njegovo ugrađeno znanje.

2.3 Okruženje

Problemi koje agenti treba da reše u nekom okruženju definišu se preko četiri parametra: učinak, okruženje, aktuatori i senzori (PEAS - performance, environment, actuators, sensors). Primer za zadatak okruženja za taksi vozača prikazan je u tabeli 2.1.

Tip agenta	Mere učinka	Okruženje	Aktuatori	Senzori
Taksi vozač	Sigurnost, brzina, poštovanje propisa, udobnost, maksimizirati dobit	Putevi, ostali učesnici u saobraćaju, pešaci, mušterija	Volan, ubrzanje, kočnica, signalizacija, sirena	Kamere, brzinometar, GPS, senzori motora, stanja goriva u rezervoaru, merač za pređeni put

Tabla 2.1

Vežba. Opisati sva četiri parametra za sledeće zadatke okruženja

- robot koji sakuplja elemente sa trake i razvrstava ih u kutije
- robot koji igra fudbal

2.3.1 Osobine okruženja

Da li se okruženje može potpuno ili parcijalno sagledati? Ako agentski senzori obezbeđuju uvid u kompletno stanje okruženja u svakom trenutku vremena onda se to okruženje može potpuno sagledati. Okruženje je potpuno sagledivo ukoliko agentski senzori detektuju sve aspekte okruženja koji su relevantni za izbor akcije. Okruženje se nekad ne može potpuno sagledati, na primer zbog buke, neadekvatnih senzora ili se nekim delovima okruženja jednostavno ne može pristupiti. Na primer, taksi vozač ne može da vidi šta misle drugi učesnici u saobraćaju. Ako agent nema senzora kažemo da se okruženje ne može uopšte sagledati.

Okruženje može imati jednog ili više agenata. Na primer, agent koji rešava ukrštene reči nalazi se u okruženju sa jednim agentom, a agent koji igra šah u okruženju sa dva agenta. Međutim, nekada se to ne može tako jednostavno odrediti, jer se postavlja pitanje koje entitete posmatramo kao agente. Na

primer, da li agent taksi vozač posmatra drugog vozača kao agenta ili kao objekat koji se ponaša prema nekim pravilima. Okruženja sa više agenata mogu biti konkurentna, kao na primer šah, a mogu biti i kooperativna u koja spada okruženje taksi vozača.

Okruženje može biti determinističko ili stohastičko. Ako je sledeće stanje okruženja potpuno određeno tekućim stanjem i akcijom koju agent izvrši tada kažemo da je okruženje determinističko, u suprotnom kažemo da je stohastičko. Okruženje usisivača je determinističko, a taksi vozača stohastičko. Za okruženje koje se ne može potpuno sagledati i koje nije determinističko kažemo da je neizvesno. Termin stohastičko obično koristimo kada nepoznavanje mogućeg ishoda izražavamo verovatnoćom, dok termin nedeterminističko koristimo za okruženja gde se samo navode mogući rezultati akcija bez uključivanja verovatnoće. Nedeterminističko okruženje se obično vezuju za situaciju kada agent treba da bude uspešan u svim mogućim slučajevima.

Za okruženje kažemo da je epizodno ako je iskustvo agenta u njemu podeljeno na epizode u kojima on prima neka zapažanja i vrši određenu akciju i akcije u jednoj epizodi ne zavise od predhodnih akcija. Primer ovakvog agenta je agent koji prikuplja delove sa trake i odlučuje koji deo će se baciti. Sa druge strane, u sekvencijalnom okruženju trenutna odluka može da utiče na sve kasnije odluke. Primer sekvencijalnih okruženja je šah i taksi vožnja.

Okruženje može biti statično ili dinamično. Ako se okruženje može promeniti dok agent odlučuje koju akciju da izvrši onda je ono dinamično, inače je statično. Lakše je delovati u statičnom okruženju jer agent ne mora stalno da posmatra svet dok razmišlja koju akciju da izvrši. Dinamičko okruženje konstanto pita agenta šta želi da uradi, ako još nije odlučio smatra se da je odlučio da ne uradi ništa. Ukoliko se u nekom periodu vremena okruženje nije promenilo, a jeste se promenio agentov učinak govorimo o poludinamičnom okruženju (na primer šah sa satom). Taksi vožnja se obavlja u dinamičnom, a rešavanje ukrštenica u statičnom okruženju.

Razlika između diskretnog i kontinualnog okruženja izražava se kroz stanje okruženja u toku vremena i zapažanja i akcije agenta. Na primer, šah ima konačan broj različitih stanja (ako nemamo sat) i ima diskretni skup zapažanja i akcija. Vožnja taksija ima kontinualne promene stanja.

Razlika između poznatog ili nepoznatog okruženja odnosi se na znanje koje agent (ili njegov kreator) ima o pravilima u okruženju. U nepoznatom okruženju agent mora da uči kako stvari funkcionišu da bi mogao da donosi prave odluke. Osobine poznatog i nepoznatog se razlikuju od toga da li se okruženje može sagledati ili ne (primer solitaire - poznato, ali se ne može potpuno sagledati, video igrice - sve se može sagledati, ali ne znamo koje komande sa tastature se koriste).

Vežba. Odrediti osobine sledećih okruženja:

- ukrštene reči

- poker

Za domaći:

- okruženje robota koji sakuplja elemente sa trake i razvrstava ih u kutije
- fudbalska utakmica

2.4 Struktura agenta

Zadatak VI je da pravi agentske programe koji implementiraju agentsku funkciju - mapiranje zapažanja na akcije. Pretpostavlja se da će se program izvršavati na nekom uređaju koji ima fizičke senzore i delove koji obavljaju određene akcije - sve ovo zajedno nazivamo arhitekturom. Program koji pravimo mora da odgovara arhitekturi (ako pravimo program koji se zove *hod* arhitektura mora da ima noge).

Agentski program

Pod agentskim programom podrazumevaćemo program čiji ulaz je zapažanje a izlaz je odgovarajuća akcija. Agentski program uzima samo trenutno zapažanje (za razliku od agentske funkcije koja uzima celokupnu istoriju zapažanja) jer ništa drugo nije dostupno iz okruženja. Ako izbor akcije zavisi od istorije zapažanja agent mora nekako da zapamti sva zapažanja.

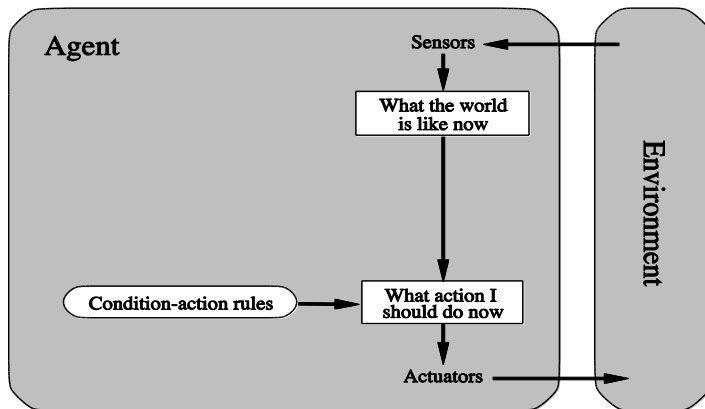
Postoje četiri tipa agentskih programa koji definišu sve osnovne principe inteligentnih sistema:

- jednostavni refleksni agenti
- refleksni agenti zasnovani na modelu
- cilj-orijentisani agenti
- korisno-orijentisani agenti

Svaki od navedinih vrsta agenata kombinuje specifične komponente na specifičan način da bi generisao odgovarajuću akciju. Svi oni se mogu transformisati u agente koji uče (learning agents).

Jednostavni refleksni agenti

Ovo je najjednostavnija vrsta agenata koja selektuje akciju na osnovu trenutnog zapažanja, ignorišući istoriju zapažanja. Primer ovakvog agenta je usisivač, ali postoje i kompleksniji primeri. Na primeru vozača taksija možemo posmatrati situaciju da ako auto ispred nas koči, odnosno pali se stop svetlo treba da zaustavimo auto. Ovakva veza se često naziva uslov-akcija ili if-else. Na slici 2.2. data je šema jednostavnog refleksnog agenta. Pravougaonicima su predstavljena trenutna unutrašnja stanja agenta, a ovalnim poljima postojeće informacije koje se koriste u odlučivanju.

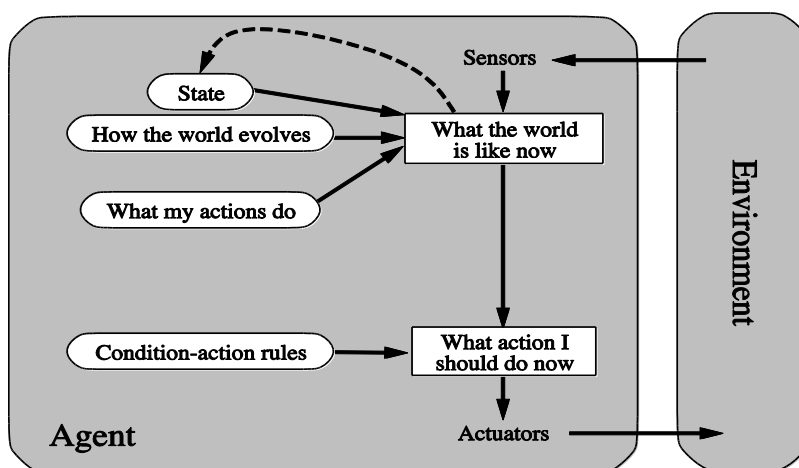


Slika 2.2 Refleksni agent

Refleksni agenti zasnovani na modelu

U okruženju koje se samo parcijalno može sagledati najbolje je pamtit i informacije o onom delu okruženja koje se trenutno ne može videti. To znači da agent treba da ima neku vrstu internog stanja koje se formira na osnovu istorije zapažanja (na primer ako taksi vozač hoće da promeni traku mora znati gde se nalazi automobil koji se možda trenutno ne vidi). Da bi se ovo interno stanje pravilno ažuriralo, dve vrste znanja se moraju ugraditi u agenta. Prvo, moramo znati kako se svet menja nezavisno od agenta (na primer, auto u susednoj traci je vremenom sve bliži). Drugo, moramo znati kako akcije agenta menjaju svet (na primer, ako se volan okrene u određenom smeru automobil će skrenuti). Znanje koje opisuje kako svet funkcioniše naziva se model, a agenti koji koriste to znanje nazivaju se agenti zasnovani na modelu.

Na slici 2.3 prikazana je struktura agenta sa internim stanjem. Ovakav agent kombinuje trenutno zapažanje sa starim internim stanjem da bi generisao novi, izmenjeni opis stanja koristeći svoje znanje o tome kako se svet menja.



Slika 2.3 Agent zasnovan na modelu

Bez obzira na to koju vrstu reprezentacije koristimo, retko kada je moguće tačno odrediti tekuće stanje u okruženju koje se ne može potpuno sagledati. Umesto toga, kućica sa labelom "What the world is like now" predstavlja agentovu najbolju moguću pretpostavku. Na primer, automatski taksi vozač ne može da vidi šta se nalazi ispred velikog kamiona koji se zaustavio ispred njega, i može samo da pretpostavi šta je uzrokovalo zastoje. Nedokučivost informacija je često neizbežna, ali agent ipak treba nešto da uradi.

Cilj-orijentisani agenti

Znati nešto o trenutnom stanju okruženja nekada nije dovoljno da se odluči šta treba uraditi. Na primer, na raskršnici puteva taksi može da skrene levo, desno ili da produži pravo. Pravilna odluka zavisi od toga gde taksi treba da stigne. Drugim rečima, pored opisa stanja, agent mora da ima i informaciju o situacijama koje su poželjne, na primer biti na destinaciji putnika koga prevozi.

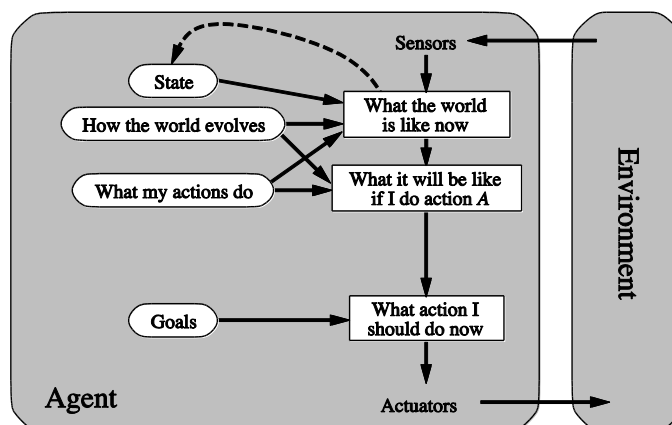
Ponekad je selekcija akcije na osnovu cilja jednoznačna, na primer kada je cilj dostižan posle jedne akcije. Češće je to mnogo komplikovanije, kada agent mora da razmotri razne opcije da bi dostigao cilj. Problemom pronalaženja niza akcija koje vode do cilja bave se dve velike oblasti VI: pretraživanje i planiranje.

Treba primetiti da se donošenje odluke na ovaj način fundamentalno razlikuje od principa uslov-akcija. Razlika je u tome što se ovde razmatra budućnost i postavljaju se pitanja tipa "Šta će se dogoditi ako uradim to i to?" i "Hoće li me to učiniti srećnim?". Kod refleksnih agenata ove informacije nisu eksplicitno predstavljene jer ugrađena pravila mapiraju direktno zapažanja na akcije. Refleksni agent će da zakoči ako vidi da se pale stop svetla vozilu ispred, dok bi cilj-orijentisani agent razmotrio da ako se vozilu ispred pali stop svetlo on će usporiti. Uzimajući u obzir znanje o tome kako svet funkcioniše, jedina akcija koja postiže cilj, a to je ne udariti vozilo ispred, je kočenje.

Iako su cilj-orijentisani agenti manje efikasni, oni su mnogo fleksibilniji jer oni svoje odluke zasnivaju na znanju koje je predstavljeno eksplicitno i može se menjati. Agent može recimo da dopuni svoje znanje o efikasnosti kočenja u slučaju kiše i da u skladu sa tim donosi ispravnije odluke. Za refleksnog agenta ovo podrazumeva pisanje i izmenu mnogih uslov-akcija pravila.

Cilj-orijentisanom agentu se može lako promeniti destinacija, postavljanjem novog cilja, dok kod refleksnog agenta jedan skup pravila uslov-akcija važi samo za jednu destinaciju, za drugu destinaciju važi potpuno drugačiji skup pravila.

Grafički prikaz cilj-orijentisanog agenta prikazan je na slici 2.4.



Slika 2.4 Cilj-orijentisani agent

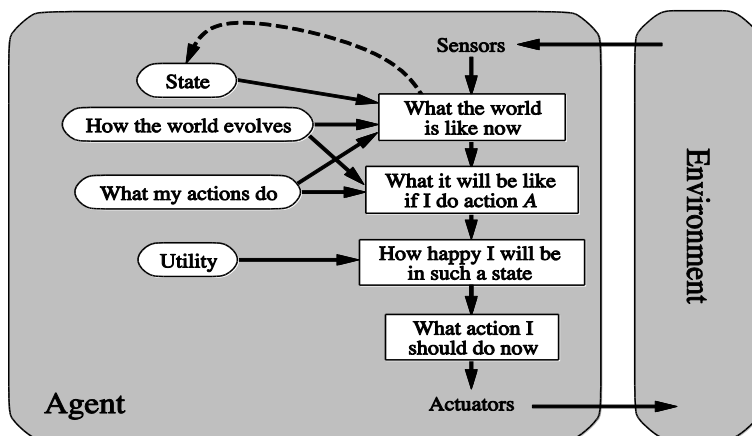
Korisno-orijentisani agenti (utility-based)

U većini okruženja ciljevi nisu dovoljni da bi se generisalo racionalno ponašanje. Na primer, postoji mnogo sekvenci akcija koje će taksistu dovesti do destinacije, ali neki od njih su brži, sigurniji, jeftiniji.

Mera učinka daje ocenu nekom nizu stanja okruženja, tako da pravi jasnu razliku između više i manje poželjnih načina da se dođe do cilja (taksi destinacije). Funkcija učinka (utility function) je interna reprezentacija mere učinka. Ako se ova interna reprezentacija i mera učinka slažu tada se agent koji bira akcije da bi maksimizirao svoj učinak (utility) može smatrati racionalnim sa tačke gledišta eksterne mere učinka.

Postoje dve vrste slučajeva kod kojih umesto cilj-orijentisanih agenata koristimo korisno-orijentisane agente. Prvi slučaj je kada imamo oprečne ciljeve i samo jedan može biti postignut (na primer brzina i sigurnost). U ovom slučaju funkcija učinka pravi neki kompromis. Drugi slučaj je kada imamo više ciljeva od kojih se ni jedan ne može sa sigurnošću dostići, tada funkcija učinka obezbeđuje da se verovatnoća uspeha iskaže u odnosu na značaj cilja.

Na slici 2.5 prikazana je struktura korisno-orijentisanog agenta. Agent koristi model sveta, zajedno sa funkcijom učinka koja iskazuje preferenciju agenta prema određenim stanjima. Na osnovu ovoga agent bira akciju koja ga vodi do najboljeg očekivanog učinka.



Slika 2.5 Korisno-orijentisani agent

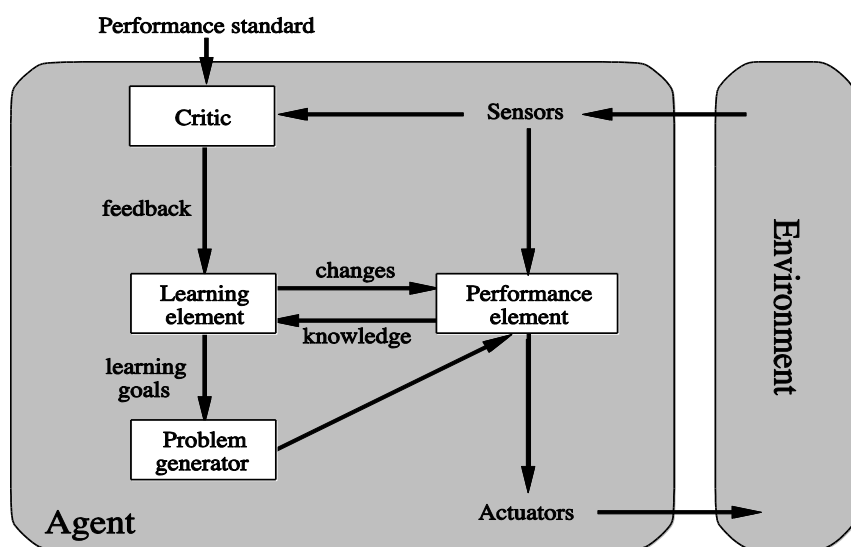
Agenti koji uče

U mnogim oblastima VI, preferira se metod koji je predložio Alan Tjuring: napraviti mašine koje umeju da uče i naučiti ih šta treba da rade.

Agenti koji uče sadrže četiri konceptualne komponente, kako je prikazano na slici 2.6. Najbitnija je razlika između elementa učenja (learning elements) i elementa izvršavanja (performance element).

Zadatak elementa učenja je da pravi poboljšanja, dok se element izvršavanja bavi selektovanjem akcija (ono što smo do sada smatrali agentom). Element učenja koristi povratne informacije od kritike (Critic) koja ocenjuje agentovo ponašanje i određuje kako se element izvršavanja može unaprediti da radi bolje. Kritika je neophodna jer zapažanja ne obezbeđuju informacije o uspehu agenta. Na primer, program koji iga šah može dobiti informaciju o tome da je matirao protivnika, međutim on mora znati da je to dobra stvar.

Poslednja komponenta je generator problema (Problem generator). Uloga ove komponente je da predlaže akcije koje će dovesti do novih i informativnih iskustava. Kroz ovu komponentu omogućeno je da agent istražuje tako što će da izvrši neke akcije koje nisu najoptimalnije kratkoročno, ali kroz njih može otkriti neke dugoročno optimalnije akcije. Uloga generatora problema je da predloži ove istraživačke akcije.



Slika 2.6 Agent koji uči

Na primeru taksi vozača element izvršavanja sadrži svo znanje i procedure koje su ugrađene u taksi vozača da bi on mogao da upravlja vozilom. Taksi vozač se kreće putevima koristeći element izvršavanja. Kritika posmatra svet i prosleđuje informaciju elementu učenja. Na primer, pošto je taksi velikom brzinom skrenuo u levo kroz tri trake, kritika registruje negativnu reakciju ostalih učesnika u saobraćaju. Na osnovu tog iskustva element učenja formuliše pravilo koje kaže da je ovakvo ponašanje neadekvatno i novo pravilo se ugrađuje u element izvršavanja. Generator problema može da identifikuje neke elemente ponašanja koje treba popraviti i da predloži eksperimente, kao što su isprobavanje kočnica na različitim podlogama i pod različitim vremenskim uslovima.

Standard performansi (Performane standard) ima ulogu da neka zapažanja rastumači kao kazne ili kao nagrade za ponašanje agenta. Na primer, ako taksi vozač nije dobio bakšiš od mušterije, i ako se to tumači kao kazna za prebrzu vožnju, agent iz toga može nešto da nauči.

Poglavlje 3

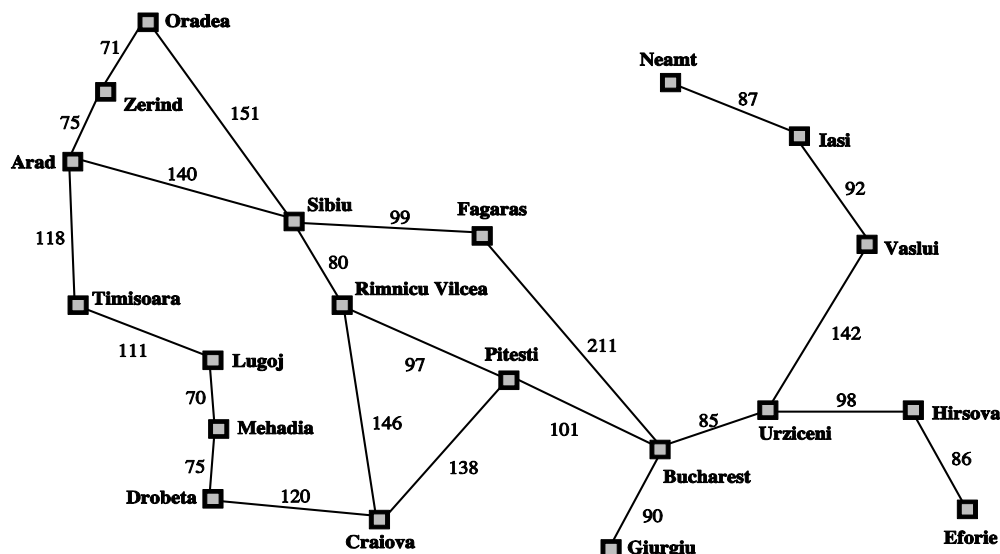
Rešavanje problema pretraživanjem

Tema ovog odeljka je jedna vrsta cilj-orijentisanih agenata koji se nazivaju agenti za rešavanje problema (problem-solving agents). Rešavanje problema započinje preciznom definicijom problema.

3.1 Agenti za rešavanje problema

Inteligenti agenti pokušavaju na maksimiziraju svoju meru učinka. Ovo je ponekad pojednostavljeno tako što agent usvaja neki cilj i nastoji da ga dostigne.

Zamislimo agenta koji se nalazi u Rumunskom gradu Aradu koji je na turističkom obilasku (Slika 3.1). Agentova mera učinka se odnosi na mnogo faktora. On pokušava da obiđe znamenitosti, da unapredi rumunski, da istraži noćni život, izbegne mamurluk, itd. Rešavanje ovakvog problema je veoma kompleksno. Umesto toga, posmatraćemo jednostavniji problem. Pretpostavimo da agent ima kartu za let iz Bukurešta narednog dana. U ovom slučaju, agent usvaja cilj da sledećeg dana bude u Bukureštu. Agentov problem je u ovom slučaju uprošćen.



Slika 3.1 Mapa Rumunije

Zadatak agenta je da odluči kako da se ponaša da bi dostigao svoj cilj. Pre toga treba da odluči koje akcije uopšte da razmatra. Da li da razmatra akcije na nivou "pomiru levu nogu nekoliko centimetara u napred" ili "okreni volan nekoliko stepeni u levo"? Posmatranjem ovakvih akcija agent ne bi uspeo ni da se isparkira u nekom razumnom vremenu, zbog toga što je na tako niskom nivou akcija potreban je veliki broj koraka da bi se dostigao cilj i postoji veliki stepen

nesigurnosti. Za ovakav tip problema dovoljno je posmatrati akcije vožnje iz jednog velikog grada u drugi. Stanja koja posmatramo su biti (nalaziti se) u nekom od velikih gradova.

Agent ima zadatak da pronađe sekvencu akcija koja ga vodi do cilja i ovaj proces se naziva pretraživanje. Algoritam za pretraživanje uzima definiciju problema kao ulaz i vraća rešenje u vidu sekvence akcija.

Pravilno definisan problem i rešenja

Problem se formalno može definisati pomoću 5 komponenti:

- Inicijalno stanje - stanje u kom se agent nalazi na početku. Na primer, u slučaju turista u Rumuniji to stanje je $In(Arad)$.
- Opis akcije koje su dostupne agentu u određenom stanju ($ACTIONS(s)$). Na primer, iz stanja $In(Arad)$ dostupne akcije su $Go(Sibiu)$, $Go(Timisoara)$, $Go(Zerind)$.
- Tranzicioni model ($Result(s,a)$)- opis rezultata akcije, odnosno šta akcija radi, kako akcija menja stanje.

Tri navedene komponente definišu prostor stanja koje formira usmereni graf u kome su stanja čvorovi, a veze između čvorova su akcije. Putanjom u grafu stanja zvaćemo bilo koji niz stanja koji je povezan nizom akcija.

Preostale dve komponente za definiciju problema su:

- Funkcija cilja - određuje da li je neko stanje cilj.
- Cena putanje - dodeljuje brojčanu vrednost svakoj putanji. Za putnika u Rumuniji to bi bio zbir pređenih kilometara. Bitan je i pojam cene koraka koja predstavlja cenu jedne akcije (razdaljina između dva susedna grada).

Rešenje problema se definiše kao niz akcija koje vode do cilja, a postoji i mera kvaliteta tog rešenja koja se izražava cenom putanje. Optimalno rešenje je ono rešenje koje ima najmanju cenu putanje.

3.2 Primeri problema

Problemi-igračke

Usisivač

Stanja: lokacija agenta i lokacija prašine, agent može biti na dve lokacije, a svaka od lokacija može biti prljava ili čista. Znači, postoji ukupno $2 \times 2^2 = 8$ stanja. Veće okruženje sa n stanja ima $n \times 2^n$ stanja.

Inicijalno stanje: bilo koje stanje

Akcije: idi levo, idi desno, usisaj

Tranzicioni model: efekti akcija su da se agent nalazi u drugom polju, zatim da je polje postalo čisto nakon akcije usisaj,... neke akcije nemaju efekta

Funkcija cilja: da li su oba polja čista

Vrednost putanje: svaki korak ima vrednost 1

8-puzzle

Stanja: lokacija praznog polja i svake od 8 pločica sa brojevima

Inicijalno stanje: bilo koje stanje može biti inicijalno stanje

Akcije: jedna akcija je pomeranje praznog polja levo, desno, gore ili dole u zavisnosti od toga gde se nalazi prazno polje

Tranzicioni model: promena položaja praznog polja i pločice koja je pomerena

Funkcija cilja: da li raspored pločica odgovara ciljnom rasporedu

Vrednost putanje: svaki korak dobija vrednost 1,

Raspored 8 kraljica na šahovskoj tabli

Rešenje ovog problema ima dva moguća pristupa. Prvi pristup je iterativna formulacija koja podrazumeva da se počinje od prazne table i u svakom koraku se dodaje nova kraljica. Drugi pristup je formulacija sa kompletnim stanjem koja počinje tako što je svih 8 kraljica na tabli i u svakom koraku se pomeraju. Formalan opis problema sa iterativnom formulacijom je:

Stanja: Bilo koji raspored 0 do 8 kraljica na tabli

Inicijalno stanje: Prazna tabla

Akcije: Dodaj kraljicu u prazno polje

Tranzicioni model: Raspored kraljica na tabli sa jednom dodatom kraljicom u određeno polje

Funkcija cilja: 8 kraljica na tabli tako da se međusobno ne napadaju

Vrednost putanje: u ovom problemu nije nam važna cena putanje već samo konačno ciljno stanje

Problemi iz realnog sveta

Pronalaženje maršrute za putovanje avionom sa presedanjem

Stanja: lokacija (aerodrom) i vreme

Početno stanje: vreme i mesto polaska putnika

Akcije: bilo koji let sa trenutne lokacije koji polazi posle trenutnog vremena, ostavljajući dovoljno vremena za ukrcavanje

Tranzicioni model: destinacija leta postaje trenutna lokacija, a vreme sletanja novo trenutno vreme

Funkcija cilja: da li se nalazimo u željenoj destinaciji

Vrednost putanje: cena leta, vreme čekanja, tip aviona, trajanje leta

Ostali primeri iz realnog sveta

Problem turističkog obilaska je sličan pronalaženju maršrute i odnosi se na problem tipa: Obići sve velike gradove u Rumuniji tačno jednom, početak i kraj obilaska je u Bukureštu. Ovde se za svako staanje, pored grada u kom se nalazimo mora pamtit i lista gradova koji smo već posetili. Sličan ovom problemu je problem trgovačkog putnika koji treba svaki grad da obiđe tačno jednom, pritom nalazeći najoptimalniju putanju.

Vežba:

1. Dati formalnu definiciju sledećeg problema:

Dato je 6 staklenih kutija u jednom redu. Svaka kutija ima bravu. Svaka od prvih 5 kutija sadrži ključ koji otvara sledeću kutiju u redu. U poslednjoj kutiji nalazi se banana. Imamo ključ od prve kutije, a želimo bananu.

2. Dati formalnu definiciju problema misionara i ljudoždera.

Tri misionara i tri ljudoždera stoje na jednoj obali reke zajedno sa čamcem koji može da prenese do dve osobe. Treba naći način da se svi prevezu na drugu obalu uz ograničenje da ni na jednom mestu ne sme biti veći broj ljudoždera od broja misionara.

3.3 Pronalaženje rešenja

Do sada smo videli kako se problemi formulišu. Sada je na redu da vidimo kako ćemo da ih rešimo. Tražimo rešenje u vidu sekvence akcija, tako da se algoritam pretraživanja svodi na razmatranje raznih mogućih sekvenci akcija. Moguće sekvence akcija koje počinju u inicijalnom stanju formiraju takozvano stablo pretraživanja čiji je koren to inicijalno stanje. Grane u stablu pretraživanja su akcije, a čvorovi su stanja iz prostoru stanja. Otvaranje (expand) čvora u stablu pretraživanja predstavlja primenu svih akcija koji su dostupni iz stanja koje je predstavljeno čvorom. Otvaranje čvora podrazumeva generisanje novih čvorova, odnosno novih stanja. Čvor koji je otvoren je čvor roditelj čvorivama koji su generisani primenom dostupnih akcija. Čvor list je čvor koji nema dece. Skup svih listova naziva se granica ili frontier (granični čvorovi). Ovaj pojam se odnosi na skup čvorova koji treba da se otvore. Proces otvaranja čvora nastavlja se dok se ne dođe do cilja ili dok više nema čvorova koji mogu da se otvore.

Način izbora čvorova za otvaranje definiše strategiju pretraživanja.

U stablu pretraživanja možemo imati slučaj da se na jednoj putanji nalaze čvorovi koji se odnose na isto stanje. Na ovaj način generišu se zatvorene putanje koje mogu da onemoguće izvršavanje nekih algoritama i da neke rešive probleme učine nerešivim.

Da bi se izbegle zatvorene putanje potrebno je zapamtiti u kojim stanjima smo bili. Postoji jedna izreka koja kaže: *Algoritam koji zaboravi svoju prošlost, prinuđen je da je ponovi.* U cilju rešavanja ovog problema za stablo pretraživanja se može vezati jedna struktura podataka koja se zove zatvorena lista (explored set ili closed list) koja pamti svaki otvoren čvor. Kada se generiše novi čvor, proverava se da li se on nalazi u ovoj listi, ukoliko postoji on se može eliminisati umesto da se doda u skup graničnih čvorova. Ovaj način pretraživanja je svojstven pretraživanju grafa pa se naziva Graph-search i on se razlikuje od verzije pretraživanja Tree-search po tome što pamti stanja u kojima je bio (Slika 3.2)


```

function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

```

```

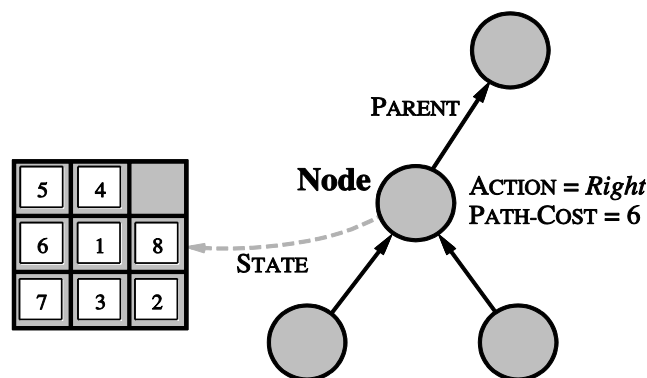
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
    if the frontier is empty then return failure
    choose a leaf node and remove it from the frontier
    if the node contains a goal state then return the corresponding solution
    add the node to the explored set
    expand the chosen node, adding the resulting nodes to the frontier
    only if not in the frontier or explored set

```

Slika 3.2

3.3.4 Strukture podataka u algoritmima za pretraživanje

Algoritmima za pretraživanje je potrebna struktura podataka za opis stabla pretraživanja. Za svaki čvor n imamo strukturu podataka koja sadrži četiri komponente (slika 3.3):



Slika 3.3 Stanje i čvor

$n.State$ - stanje iz prostora stanja

$n.Parent$ - čvor u stablu pretraživanja koji je generisao čvor n

$n.Action$ - akcija kojom je generisan čvor n

n.Path-Cost - cena putanje, obeležava se i sa $g(n)$.

Koja je razlika između čvora i stanja? Čvor je struktura podataka u stablu pretraživanja, a stanje se odnosi na osobine okruženja. Jedan čvor se može odnositi na više stanja, ako je stanje generisano na različitim putanjama u stablu pretraživanja.

Granični čvorovi se moraju organizovati na specifičan način kako bi algoritam za pretraživanje jednostavno mogao da izabere sledeći čvor za otvaranje (ekspandovanje). Najprimerenija struktura podataka za ovo je red (queue). Postoje različite vrste redova koje se razlikuju prema redosledu dodavanja i skidanja elemenata. Tri najčešće varijante su:

- FIFO red – first in first out
- LIFO red – last in first out (stek)
- red prioriteta – koji bira element sa najvećim prioritetom prema nekoj funkciji poretka.

3.3.5 Merenje performansi algoritama za pretraživanje

Posmatraju se četiri osobine:

- kompletnost - da li će algoritam sigurno pronaći rešenje ako ono postoji?
- optimalnost - da li će pronaći optimalno rešenje?
- vremenska složenost - koliko vremena je potrebno da se pronađe rešenje?
- prostorna složenost - koliko memorije je potrebno za pretraživanje?

Vremenska i prostorna složenost se razmatraju u odnosu na neku meru složenosti problema. U teoriji računarstva uobičajen mera je veličina grafa stanja koja se izražava izrazom $|V| + |E|$ gde je V skup čvorova (vertices) a E skup grana (edges). Ovakva je mera je moguća kod konačnih eksplicitno predstavljenih grafova. U VI je stablo pretraživanja predstavljeno implicitno preko inicijalnog stanja, akcija i tranzicionog modela i često je beskonačno. Zbog svega ovoga za izražavanje složenosti algoritama za pretraživanje u VI koriste se sledeći elementi:

- b - faktor grananja (maksimalan broj naslednika),
- d - dubina najplićeg cilja (broj koraka do korena do cilja koji jemu je najbliži),
- m - maksimalna dužina bilo koje putanje grafa stanja.

Vremenska složenost se obično odnosi na broj čvorova koji se generišu tokom pretraživanja, a prostorna složenost preko broja čvorova koji se čuvaju u memoriji.

3.4 Strategije za slepo pretraživanje

Slepo ili neinformisano pretraživanje odnosi se na strategije kod kojih ne postoji dodatno znanje o stanjima osim onog koje je dato u definiciji problema. Kod ove vrste pretraživanja moguće je samo generisati potomke u stablu i utvrditi da li je neko stanje cilj ili nije. Strategija pretraživanja određena je redosledom otvaranja graničnih čvorova i razlikujemo 5 strategija:

- pretraživanje prvo u širinu
- uniform-cost search
- pretraživanje prvo u dubinu
- ograničeno pretraživanje u dubinu
- pretraživanje prvo u dubinu sa iterativnim povećanjem dubine

Strategije koje imaju informaciju o tome koje stanje više „obećava“ nazivaju se informisano ili heurističko pretraživanje i ono je tema odeljka 3.5.

3.4.1. Pretraživanje prvo u širinu

Kod pretraživanja prvo u širinu prvo se otvara koren, zatim njegovi potomci, zatim potomci njegovih potamaka, itd. Prvo se otvaraju svi čvorovi na jednoj dubini, pre nego što se otvore čvorovi na sledećoj.

Pretraživanjem prvo u širinu otvara se čvor koji je najbliži korenu što se postiže korišćenjem FIFO reda za skladištenje graničnih čvorova. Na slici 3.4 prikazan je pseudo kod algoritma za pretraživanje prvo u širinu.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier) /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Slika 3.4 Algoritam za pretraživanje prvo u širinu

Kakve su performanse ovog algoritma? On je kompletan jer ukoliko je najplići čvor koji predstavlja ciljno stanje na nekoj konačnoj dubini d , on će sigurno biti pronađen nakon što se otvore svi plići čvorovi (pretpostavljamo da je faktor grananja b konačan). Kada se generiše ciljni čvor to je sigurno najplići ciljni čvor koji je generisan, međutim to ne mora da znači da je on i najoptimalniji. Tehnički, pretraživanje prvo u širinu je optimalno ako je cena putanje neopadajuća funkcija dubine čvora. Najčešći slučaj ovakvog scenarija je kada svaka akcija ima istu cenu.

Vremenska i prostorna složenost nisu tako sjajne za ovu strategiju pretraživanja. Pretpostavimo da svaki čvor ima b predaka. Koren generiše b čvorova, zatim svako od njih po b čvorova,...I pretpostavimo da je rešenje na dubini b , u najgorem slučaju to je poslednji čvor generisan na toj dubini, tada je ukupan broj generisanih čvorova (vremenska složenost):

$$b + b^2 + b^3 + \dots + b^d = O(b^d)$$

U slučaju da algoritam poziva funkciju cilja kada se čvor selektuje za otvaranje umesto kada se izgeneriše, svi čvorovi na dubini d biće otvoreni pre nego što se pronađe cilj, te je u tom slučaju vremenska složenost $O(b^{d+1})$.

Što se tiče prostorne složenosti, ova strategija pretraživanja sve generisane čvorove drži u memoriji, što znači da je i prostorna složenost $O(b^d)$.

Zaključak je da je ovaj algoritam izuzetno zahtevan memorijski i vremenski.

3.4.2 Pretraživanje vođeno cenom putanje (Uniform-cost search)

Ova strategija pretraživanja otvara čvor n sa najmanjom cenom putanje $g(n)$. Ovo se realizuje korišćenjem reda prioriteta za skladištenje graničnih čvorova. Ukoliko svaki potez ima jednaku cenu ova strategija se svodi na pretraživanje prvo u širinu.

Na slici 3.5 prikazan je pseudo kod algoritma za pretraživanje vođeno cenom putanje.

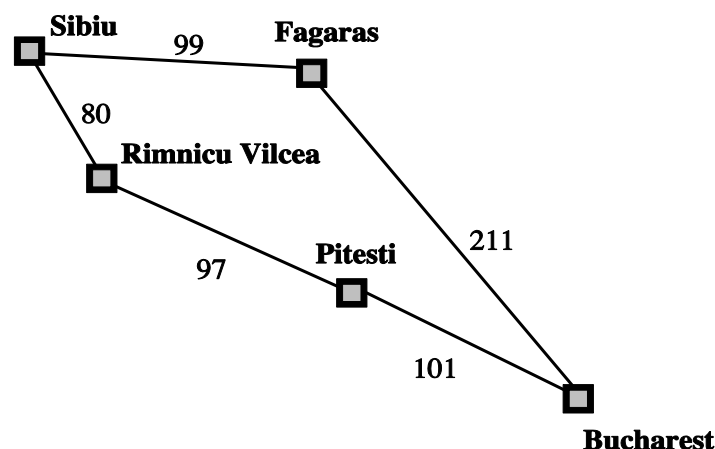
```

function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
  node  $\leftarrow$  a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  frontier  $\leftarrow$  a priority queue ordered by PATH-COST, with node as the only element
  explored  $\leftarrow$  an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node  $\leftarrow$  POP(frontier) /* chooses the lowest-cost node in frontier */
    if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child  $\leftarrow$  CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        frontier  $\leftarrow$  INSERT(child, frontier)
      else if child.STATE is in frontier with higher PATH-COST then
        replace that frontier node with child

```

Slika 3.5 Algoritam za pretraživanje vođeno cenom putanje

Kod ove strategije pretraživanja funkcija cilja se poziva kada se čvor selektuje za otvaranje, a ne kada se generiše. Razlog za ovo je što prvi generisani ciljni čvor ne mora biti optimalno rešenje. Druga specifična stvar kod ovog algoritma je da se izvršava test koji za dva ista granična čvora proverava koja je putanja optimalnija. Primer za ovaj slučaj je problem dolaska iz grada Sibiu u Bucharest (slika 3.6).



Slika 3.6 Deo mape Rumunije

Potomci Sibiu-a u stablu pretraživanja su Rimnicu Vilcea i Fagaras sa cenama 80 i 99. Čvor koji ima manju cenu je Rimnicu Vilcea i on se sledeći otvara. Dodaje se Pitesti i sada je cena putanje $80+97 = 177$. Čvor sa najmanjom cenom je sada Fagaras i on se otvara i dodaje

Bukurešt na putanju čija je cena $99+211=310$. Iako je generisan ciljni čvor ali algoritam nastavlja pretraživanje i otvara Pitesti dodajući Bukurešt sa cenom putanje $80+97+101=278$. Sada algoritam proverava da li je ova putanja bolja od prethodno generisane i pošto jeste prethodna putanja se odbacuje. Sada se selektuje Bucharest za otvaranje i funkcija cilja vraća vrednost true.

Kakve su performanse ovog algoritma? On je optimalan u opštem slučaju. Možemo reći da kada algoritam pretraživanja vođenim cenom putanje selektuje ciljni čvor G za otvaranje imamo optimalnu putanju do njega. Da ovo nije slučaj postojao bi granični čvor n' koji se nalazi na još neotvorenoj putanji koji od korena vodi do nekog optimalnog cilja G_1 . Po definiciji algoritma, n' ima manju cenu putanje od G_1 , pošto je put do G_1 optimalan, a do G nije sledi da je $g(G_1) \leq g(G)$, odnosno $g(n') \leq g(G)$. To znači da bi čvor n' bio selektovan za otvaranje pre ciljnog čvora G .

Algoritam je kompletan ako pretpostavimo da je cena svakog koraka veća od neke male pozitivne konstante ξ . U suprotnom, ako bi postojala putanja sa akcijama čija je cena 0, algoritam bi se zaglavio u beskonačnoj petlji.

Algoritam je vođen cenom putanje, a ne dubinom, zbog čega je teško izraziti njegovu složenost preko parametara b i d . Umesto toga koristimo vrednost C^* koja predstavlja cenu putanje do optimalnog rešenja i ξ koja predstavlja minimalnu moguću cenu jednog koraka. Sada, najgori slučaj vremenske i prostorne složenosti je $O(b^{\lfloor C^*/\xi \rfloor})$ što može biti mnogo veće od b^d . To je zbog toga što algoritam može da istražuje veliko stablo koje se sastoji od malih koraka, pre nego što počne da istražuje deo stabla koji obuhvata veće i verovatno korisnije korake. Kada su sve cene koraka jednake onda je $O(b^{\lfloor C^*/\xi \rfloor})$ jednako b^{d+1} .

3.4.3 Pretraživanje prvo u dubinu

Algoritam pretraživanja prvo u dubinu otvara prvo čvor koji je najdalji korenu. Algoritam se odmah spušta do najdubljeg nivoa stabla pretraživanja i zaustavlja se kada čvor više nema potomaka. Kada dođe do čvora koji nema potomaka, taj čvor se izbacije iz skupa graničnih čvorova i algoritam se vraća na sledeći čvor sa prethodnog nivoa.

Pretraživanje prvo u dubinu koristi stek (LIFO red) za skladištenje graničnih čvorova što znači da se poslednji izgenerisan čvor bira za otvaranje.

Kakve su performanse ovog algoritma? On nije kompletan u beskonačnom prostoru stanja jer može da upadne u beskonačnu putanju koja ne vodi do cilja. Iz sličnog razloga algoritam nije optimalan. Ukoliko je recimo čvor A korenski čvor i on ima dva potomka B (levi potomak) i C (desni potomak) i pretpostavimo da je C ciljni čvor. Algoritam će pretražiti celo levo podstablo pre nego što dođe do čvora C . U tom podstablu se može naći neki drugi ciljni čvor na koga će algoritam prvo naići iako to nije optimalno rešenje.

Algoritam može generisati svih $O(b^m)$ čvorova gde je m maksimalna dubina bilo kog čvora. Treba primetiti da m može biti mnogo veće od d (dubina najplićeg ciljnog čvora), a može biti i beskonačno ako imamo neograničeno stablo pretraživanja.

Jedina prednost ovog algoritma u odnosu na prethodne je prostorna složenost. Pretraživanje prvo u dubinu u memoriji drži samo jednu putanju od korena do lista zajedno sa neotvorenim rođacima. Kada se čvor otvori, i ispituju se svi njegovi potomci on se briše iz memorije. Za prostor stanja sa faktorom grananja b i maksimalnom dubinom m pretraživanje prvo u dubinu ima zauzeće memorije od $O(bm)$ čvorova.

3.4.4 Pretraživanje sa ograničenom dubinom

Određeni nedostaci pretraživanja prvo u dubinu mogu se prevazići uvođenjem ograničenja na dubinu pretraživanja (maksimalnu dubinu stabla pretraživanja označavamo sa l). Čvorovi na dubini l se tretiraju kao da nemaju potomaka. Ovakva vrsta algoritma naziva se pretraživanje sa ograničenom dubinom. Opisani pristup rešava problem beskonačnih putanja, ali ostaje problem nekompletnosti, ako izaberemo $l < d$, što znači da je ciljni čvor ispod granice na dubinu. Pretraživanje sa ograničenom dubinom nije optimalno ako je $l > d$. Vremenska složenost ovog algoritma je $O(b^l)$, a prostorna složenost je $O(b \cdot l)$. Pretraživanje prvo u dubinu je specijalan slučaj pretraživanja za ograničenom dubinom za $l = \infty$.

Ograničenje na dubinu se može definisati na osnovu znanja o problemu. Na primer, na mapi Rumunije ima 20 gradova što znači da ako postoji rešenje problema ono mora biti na maksimalnoj dubini 19 što znači da za ovaj problem možemo koristiti pretraživanje sa ograničenom dubinom sa $l = 19$. Međutim, ako malo bolje prostudiramo mapu videćemo da se iz svakog grada može doći do bilo kog grada u maksimalno 9 koraka. Ovaj broj, poznat i kao prečnik (diameter) prostora stanja predstavlja bolji izbor limita pretraživanja. Međutim, za većinu problema ne možemo znati koji je limit odgovarajuća.

3.4.5 Pretraživanje prvo u dubinu sa iterativnim povećavanjem dubine

Pretraživanje sa iterativnim povećavanjem dubine je strategija koja se koristi sa pretraživanjem prvo u dubinu i pronalazi najbolji limit pretraživanja. Ovo se postiže iterativnim povećavanjem limita koji je na početku 0, zatim 1, 2, i tako dalje sve dok se ne dostigne cilj. Cilj će se dostići na limitu koji je jednak d (dubina najplićeg cilja).

Ovaj algoritam pretraživanja koristi prednosti pretraživanja prvo u dubinu i pretraživanja prvo u širinu. Kao kod pretraživanja prvo u dubinu, njegovi memorijski zahtevi su skromni - $O(bd)$. A slično pretraživanju prvo u širinu on je kompletan ako je faktor grananja konačan i optimalan ako je cena putanje neopadajuća funkcija dubine čvora.

Pretraživanje sa iterativnim povećavanjem dubine se može učiniti kao bespotrebno trošenje vremena na generisanje čvorova više puta. Međutim, ispostavlja se da ovo nije previše skupo jer u stablima sa istim (ili sličnim) faktorom grananja na svakom nivou, većina čvorova se nalazi na donjem nivou, tako da i nije toliko važno što se čvorovi na višim nivoima generišu više puta. U ovom algoritmu čvorovi na donjem nivou (na dubini d) generišu se jednom, na dubini $d-1$ dva puta, i tako dalje sve do dece korena koja se generišu d puta. Znači da je ukupan broj generisanih čvorova u najgorem slučaju:

$$(d)b + (d-1)b^2 + \dots + (1)b^d,$$

što daje vremensku složenost od $O(b^d)$ kao i kod pretraživanja prvo u širinu.

Pretraživanje sa iterativnim povećavanjem dubine se smatra najpogodnijom neinformisanom strategijom pretraživanja kada je prostor stanja veliki i dubina rešenja nije poznata.

3.4.6 Dvosmerno pretraživanje

Ideja dvosmernog pretraživanja je da se dva pretraživanja pokrenu istovremeno, jedno od inicijalnog stanja i jedno od cilja, sa nadom da će se sresti u sredini. Motivacija za ovo je složenost od

$$b^{d/2} + b^{d/2} < b^d$$

Dvosmerno pretraživanje se implementira tako što se funkcija cilja zamenjuje funkcijom koja proverava da li se seku skupovi graničnih čvorova ova dva pretraživanja. Prvo pronađeno rešenje ne mora biti optimalno.

Osnovni problem kod ove vrste pretraživanja je što je često veoma teško napraviti pretraživanje koji počinje od cilja.

3.5 Startegije za informisano (heurističko) pretraživanje

Pored same definicije problema, informisano pretraživanje koristi i znanje o problemu. Zahvaljujući tom znanju ova vrsta pretraživanja je efikasnija od neinformisanog ili slepog pretraživanja.

Uopšteni pristup informisanog pretraživanja naziva se „prvo najbolje“ u kom se selekcija čvora za otvaranje vrši na osnovu funkcije evaluacija $f(n)$. Funkcija evaluacije predstavlja procenu cene (troškova) iz čvora n i za otvaranje se selektuje čvor sa najmanjom vrednošću funkcije evaluacije. Implementacija pretraživanja „prvo najbolje“ slična je implementaciji pretraživanja vođenim cenom putanje (odjeljak 3.4.2) osim što se red prioriteta graničnih čvorova kreira preko $f(n)$, a ne $g(n)$.

Kod informisanog pretraživanja strategije se razlikuju na osnovu izbora funkcije evaluacije ($f(n)$). Većina pretraživanja „prvo najbolje“ za formiranje funkcije evaluacije koristi heurističku funkciju koja se obeležava sa $h(n)$ i definiše na sledeći način:

$h(n)$ - procenjena cena najpovoljnije putanje od stanja u čvoru n do ciljnog stanja.

Jedan primer heurističke funkcije za turistu u Rumuniji je procena najpovoljnije putanje od Arada do Bukurešta preko prave linije (vazdušne linije) koja vodi od Arada do Bukurešta.

Heuristička funkcija je najčešće primenjeni način da se znanje o problemu iskoristi u algoritmu za pretraživanje. O heurističkoj funkciji će biti više reči kasnije, za sada ćemo je smatrati proizvoljnom, nenegativnom funkcijom koja je vezana za problem koji se rešava pretraživanjem, sa jednim ograničenjem, ukoliko čvor n predstavlja ciljno stanje onda je $h(n)=0$. Postoje dva najčešća načina za korišćenje heurističke funkcije u pretraživanju, to su pohlepno pretraživanje prvo najbolje i A^* pretraživanje.

3.5.1 Pohlepno pretraživanje prvo najbolje

Pohlepno pretraživanje prvo najbolje pokušava da otvori čvorove koji su najbliži cilju sa petpostavkom da će na ovaj način brzo doći do rešenja. Ovaj algoritam procenjuje čvor samo na osnovu vrednosti heurističke funkcije, odnosno $f(n)=h(n)$.

Razmotrimo ovu strategiju pretraživanja na primeru turiste u Rumuniji. Za heurističku funkciju ćemo uzeti razdaljinu vazdušnom linijom (straight-line distance) koju ćemo obeležiti sa h_{SLD} . Ako je cilj stići do Bukurešta moramo znati udaljenost svakog grada od Bukurešta vazdušnom linijom (slika 3.7).

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

Slika 3.7

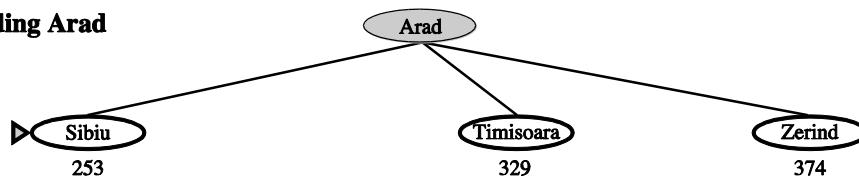
Na slici 3.8 prikazano je stablo pretraživanja za pronalazak putanje od Arada do Bukurešta. Koristi se heuristička funkcija razdaljine vazdušnom linijom (Slika 3.7).

Algoritam koji na ovaj način pronalazi rešenje nije optimalan. Putanja koju je on pronašao preko Sibiu i Fagarasa nije najkraća, ona je 32 kilometra duža od putanje preko Rimnicu Vilcea i Pitesti. Ovo pokazuje zašto se algoritam zove „pohlepan“ – u svakom koraku on pokušava da bude što bliže cilju.

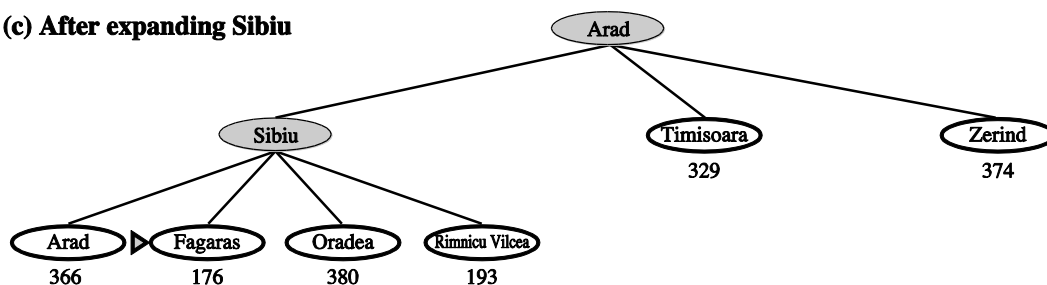
(a) The initial state



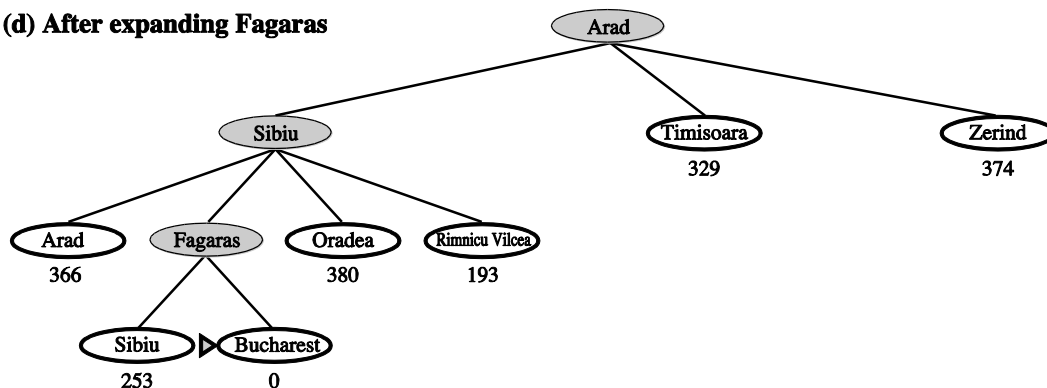
(b) After expanding Arad



(c) After expanding Sibiu



(d) After expanding Fagaras



Slika 3.8

Ovaj algoritam je takođe i nekompletan, čak i u konačnom prostoru stanju. Ako posmatramo na primer problem putovanja od Iasi do Fagarasa, videćemo da heuristička funkcija predlaže da se izabere Neamt jer je on bliži Fagarasau, međutim to je kraj puta. Kada se otvori Neamt Iasi je ponovo u skupu graničnih čvorova i ima manju heurističku funkciju od Vasluija koji je takođe granični čvor. Znači Iasi će biti ponovo selektovan za otvaranje i tako ulazimo u beskonačnu petlju. U najgorem slučaju prostorna složenost je $O(b^m)$ gde je m maksimalna dubina stabla pretraživanja. Sa dobrom heurističkom funkcijom, ova složenost bi se mogla značajno smanjiti.

3.5.3 A* pretraživanje

Najpoznatija strategija pretraživanja prvo najbolje naziva se A^* (A-star) pretraživanje. Ova strategija procenjuje čvor kombinujući $g(n)$ i $h(n)$, odnosno funkcija evaluacije je

$$f(n) = g(n) + h(n).$$

Pošto je $g(n)$ cena putanje od startnog čvora do čvora n , a $h(n)$ je procenjena cena najpovoljnije putanje od n do cilja, onda za $f(n)$ možemo reći da je to procenjena cena najpovoljnijeg rešenja koje prolazi kroz n . Da bi A^* pretraživanje bilo optimalno i kompletno heuristička funkcija mora da zadovolji određene uslove.

Uslovi za optimalnost: dopustivost i monotonost

Prvi uslov koji mora biti zadovoljen je da je heuristička funkcija dopustiva. Dopustiva heuristička funkcija nikada ne precenjuje cenu dostizanja cilja (dodeljuje vrednost manju ili jednaku stvarnoj ceni). Pošto je $g(n)$ stvarna cena putanje od korena do čvora n , tada možemo reći da $f(n)$ ne precenjuje stvarnu cenu putanje od korena do rešenja preko čvora n .

Dopustiva heuristika je optimistična zato što ona procenjuje cenu rešavanja problema manjom nego što ona stvarno jeste. Jedan od primera dopustive heuristike je udaljenost između gradova vazdušnom linijom, jer je prava linija sigurno najmanja razdaljina između gradova, tako da ta vrednost nije precenjena. Na slici 3.9 prikazano je stablo A^* pretraživanja koje traži najpovoljniji put od Arada do Bukurešta koristeći heuristiku udaljenosti vazdušnom linijom.

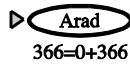
Drugi, jači uslov za optimalnost A^* pretraživanja (u verziji pretraživanja grafa) je monotonost. Heuristička funkcija $h(n)$ je monotona ako za svaki čvor n i svaki njegov potomak n' generisan nekom akcijom a , procenjena cena dostizanja cilja iz n nije veća od cene koraka akcije a i cene dostizanja cilja iz n' , odnosno

$$h(n) \leq c(n, a, n') + h(n')$$

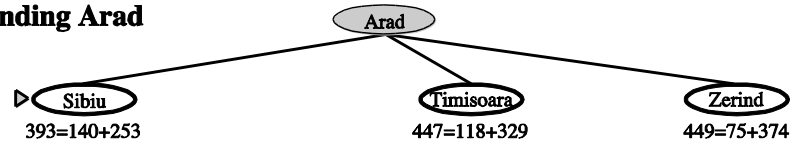
Ovo je jedan oblik nejednakosti trougla (dužina hipotenuze nije veća od zbira dužina kateta), pri čemu trougao čine n , n' i ciljni čvor koji je najbliži n .

Može se pokazati da je svaka monotona heuristika i dopustiva, što znači da je monotonost jači uslov od dopustivosti. Vrlo je teško naći heuristiku koja je dopustiva, a nije monotona. Heuristika definisana kao razdaljina između gradova vazdušnom linijom je monotona.

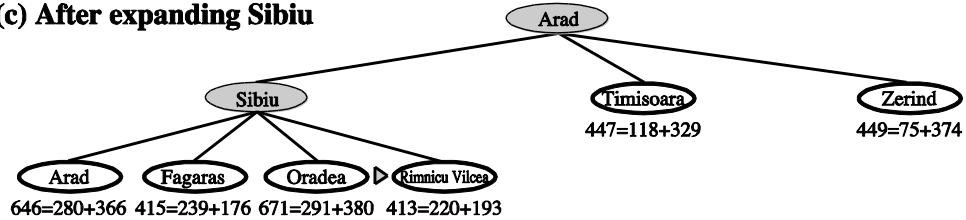
(a) The initial state



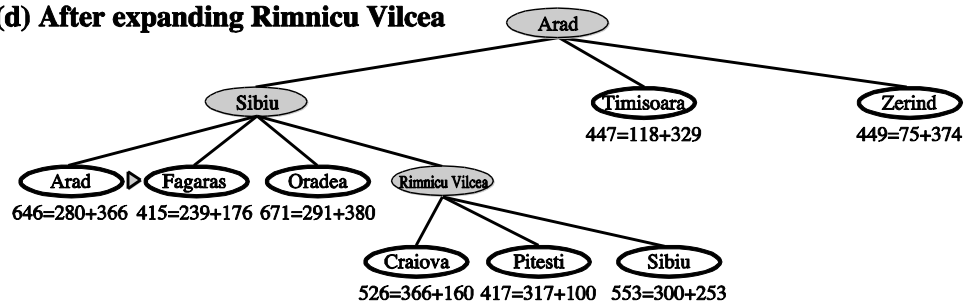
(b) After expanding Arad



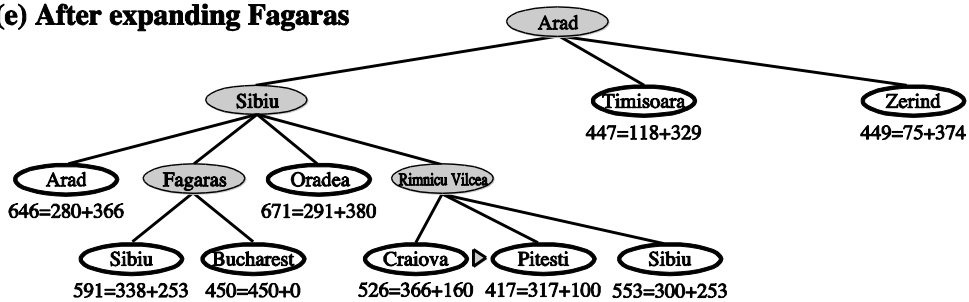
(c) After expanding Sibiu



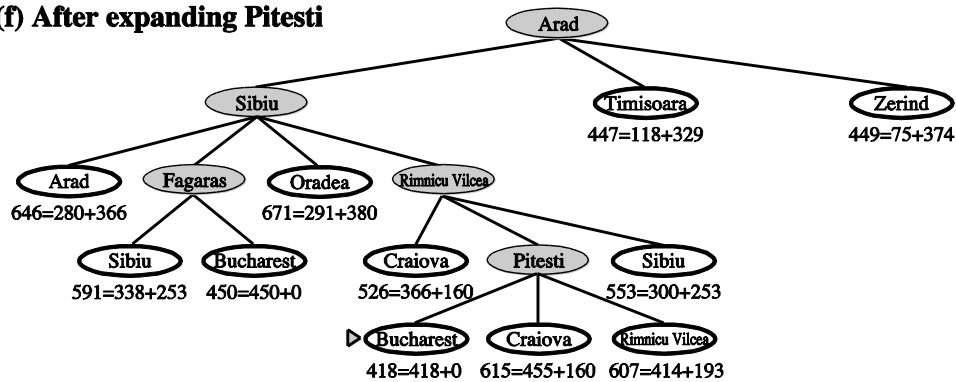
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras



(f) After expanding Pitesti



Slika 3.9

Optimalnost A* pretraživanja

A* pretraživanje ima sledeće osobine:

tree-search verzija A* pretraživanja je optimalna ako je $h(n)$ dopustiva, a graph-search verzija je optimalna ako je $h(n)$ monotona.

Dokazaćemo drugu tvrdnju. Prvi korak je da se utvrdi sledeće: ako je $h(n)$ monotona, tada su vrednosti $f(n)$ neopadajuće dužinom putanje. Dokaz za ovo proizilazi iz definicije monotonosti. Pretpostavimo da je n' potomak od n , tada je $g(n')=g(n)+c(n,a,n')$ za neku akciju a , i tada imamo:

$$f(n')=g(n')+h(n')=g(n)+c(n,a,n')+h(n')\geq g(n)+h(n)=f(n).$$

Poslednju nejednakost dobijemo kao posledicu monotonosti $h(n)$.

Sledeće što treba dokazati je da kada A* selektuje ciljni čvor G za otvaranje nađena je optimalna putanja do tog čvora. Ukoliko ovo nije slučaj, postoji drugi ciljni čvor G' i postoji granični čvor n koji je na optimalnoj putanji od startnog čvora do G' . Kako je f neopadajuća funkcija duž jedne putanje važi sledeće

$$f(n)\leq f(G')=g(G') \quad \text{jer je } h(G')=0$$

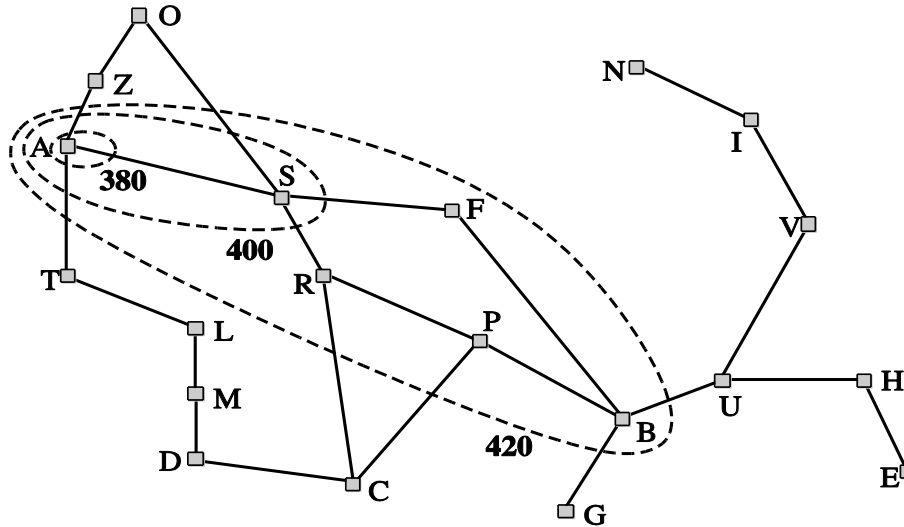
$$< g(G) \quad \text{jer je } G' \text{ optimalno a } G \text{ nije}$$

$$= f(G) \quad \text{jer je } h(G)=0$$

Sledi da će n biti selektovan pre G što je kontradikcija jer je G već izabran za selektovanje.

Činjenica da je f neopadajuća funkcija duž bilo koje putanje znači da možemo da nacrtamo konture u prostoru stanja, kao na topografskoj mapi. Slika 3.8 pokazuje primer kontura. Unutar konture koja je obeležena sa 400, svi čvorovi imaju vrednost $f(n)$ manju ili jednaku sa 400. Kako A* otvara granične čvorove sa najmanjom vrednosti funkcije, možemo videti kako se A* širi oko početnog čvora, dodajući čvorove iz koncentrične krivulje koja opisuje rastuću funkciju f .

Sa pretraživanjem zasnovanim na ceni putanje (uniform cost search) koja je ekvivalentna A* pretraživanju za $h(n)=0$, ove krivulje imaju oblik krugova oko početnog čvora. Sa što tačnijom heuristikom dobijamo krivulje koje što uže obuhvataju optimalnu putanju.



Slika 3.10

Ako je C^* cena putanje do optimalnog cilja tada možemo reći sledeće:

- A^* otvara sve čvorove za koje je $f(n) < C^*$.
- A^* može, posle toga da otvori neke od čvorova koji su tačno na "ciljnoj konturi" (gde je $f(n) = C^*$) pre nego što selektuje ciljni čvor.

Kompletnost A^* pretraživanja podrazumeva da postoji konačno mnogo čvorova sa cenom manjom ili jednakom C^* . Ovaj uslov je ispunjen ako je cena svakog koraka veća od nekog konačnog ξ i faktor grananja b je konačan.

Treba primetiti da A^* ne otvara čvorove za koje važi da je $f(n) > C^*$, na primer Timisoara nikada neće biti otvorena (Slika 3.9) iako je dete korena. Kažemo da je podstablo čiji je koren Timisoara odsečeno. Pošto je h_{SLD} dopustiva, algoritam može slobodno da ignoriše ovo podstablu a da se ne naruši optimalnost. Koncept odsecanja, odnosno eliminacije nekih mogućnosti iz razmatranja, je važan u mnogim oblastima VI.

Za A^* algoritam možemo reći da je optimalno efikasan što znači da nijedan drugi optimalan algoritam neće otvoriti manje čvorova od A^* . To je zbog toga što algoritam koji ne otvori sve čvorove za koje važi da je $f(n) < C^*$ rizikuje da promaši optimalno rešenje.

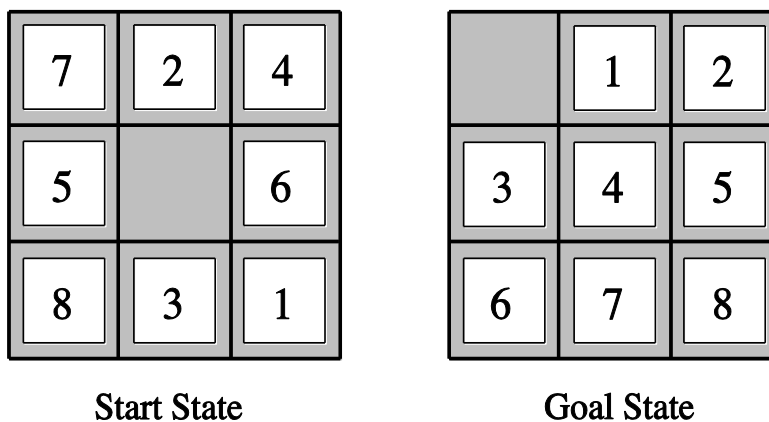
Iako zvuči savršeno A^* algoritam ne daje odgovor na sve probleme pretraživanja. Problem je u tome što za većinu problema broj stanja u ciljnoj konturi eksponencijalan u odnosu na dužinu ciljne putanje. Prostorna složenost ovog algoritma je prevelika tako da je on neupotrebljiv za mnoge kompleksne probleme.

ostoje brojne verzije algoritama koji prevazilaze prostornu kompleksnost bez ugrožavanja osobina kompletnosti i optimalnosti. Neki od njih su A* pretraživanje sa iterativnim povećavanjem dubine, rekurzivno pretraživanje prvo najbolje, memorijski ograničeno A* pretraživanje (MA* – memory-bounded A*) i pojednostavljeno memorijski ograničeno A* (simplified MA*) pretraživanje.

Da li može agent da nauči da pretražuje bolje? Odgovor je da, i metod učenja agenta zasniva se na konceptu koji se naziva meta-nivo prostora stanja. Svako stanje u metanivou prostora stanja oslikava interno stanje programa koji implementira pretraživanje u objektnom-nivou prostora stanja (kao što je Rumunija). Na primer, interno stanje A* pretraživanja sadrži trenutno stablo pretraživanja, odnosno jedno stanje u metanivou prostora stanja sadrži jedno stanje stabla pretraživanja, a otvaranje lista sadrži prelazak iz jednog stanja u metanivou u drugo. Učenje na metanivou može da uoči neka podstabla koje mogu da se izostave u pretraživanju.

3.6 Heurističke funkcije

Analiziraćemo problem izbora odgovarajuće heurističke funkcije na primeru rešavanja slagalice pločica (8-puzzle)(Slika 3.11). Rešavanje ovog problema sastoji se od pomeranja pločica horizontalno i vertikalno u prazno polje sve dok se pločice ne postave u odgovarajući redosled.



Slika 3.11

Prosečno rešenje ovog problema sastoji se od 22 koraka. Faktor grananja je u proseku 3 (Kada je prazno polje u sredini, imamo 4 moguća poteza, ako je u ćošku onda imamo 2, ako je uz ivicu imamo 3 moguća poteza). Ovo znači da bi stablo pretraživanja za neinformisane strategije imalo oko $3^{22} \approx 3.1 \times 10^{10}$ čvorova. Graf verzija pretraživanja bi smanjila ovaj broj na oko 170000 jer je samo $9!/2 = 181.440$ različitih stanja dostižno (postoje dva različita ciljna stanja koja dele skup mogućih rasporeda pločica na 2 dela, dokazati!).

Ovi brojevi i nisu toliko veliki, međutim za verziju 15 pločica broj čvorova je 10^{13} , što nameće pitanje dobre heurističke funkcije koja bi smanjila ovaj broj stanja. Postoji veliki broj razmatranih heuristika, a dve su najpoznatije

- h_1 - broj pločica koje nisu na svom mestu, za slučaj na slici 3.11 $h_1=8$, ovo je svakako dopustiva heuristika, jer je jasno da svaka pločica koja nije na svom mestu mora da bude pomerena makar jednom.

- h_2 - zbir udaljenosti pojedinačnih pločica od ciljne pozicije (Manhetn udaljenost). Kako pločice ne mogu da se pomeraju po dijagonali, udaljenost će se meriti kao suma vertikalne i horizontalne udaljenosti. Ova heuristika je takođe dopustiva jer je najbolje što u jednom potezu može da se uradi da se pomerimo jedan korak bliže cilju. Za primer na slici 3.11 h_2 iznosi $3+1+2+2+2+3+3+2=18$.

Nijedna od ove dve heuristike ne precenjuje stvarnu cenu koja iznosi 26.

3.6.1 Kvalitet heurističke funkcije

Jedan od načina da se okarakteriše kvalitet heurističke funkcije je efektivni faktor grananja (b^*). Ako je ukupan broj čvorova generisanih A^* pretraživanjem jednak N i dubina rešenja je d , tada je b^* faktor grananja koje treba da ima stablo pretraživanja dubine d da bi imalo N čvorova.

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Na primer, ukoliko A^* pronalazi rešenje na dubini 5 i otvori 52 čvora, tada je efektivni faktor grananja 1.92.

Dobro dizajnirane heurističke funkcije će imati efektivni faktor grananja blizu 1.

Na slici 3.12 prikazano je poređenje algoritama pretraživanja u 1200 proizvoljno izgenerisanih situacija u igri 8-puzzle u kojima se do rešenja dolazi u 2 do 24 koraka. Posmatraju se tri verzije algoritma: iterativno povećavanje dubine, pretraživanje sa heuristikom h_1 i h_2 i dva parametra, to su prosečan broj izgenerisanih čvorova i prosečan efektivni faktor grananja.

d	Search Cost			Effective Branching Factor		
	IDS	$A^*(h_1)$	$A^*(h_2)$	IDS	$A^*(h_1)$	$A^*(h_2)$
2	10	6	6	2.45	1.79	1.79
4	112	13	12	2.87	1.48	1.45
6	680	20	18	2.73	1.34	1.30
8	6384	39	25	2.80	1.33	1.24
10	47127	93	39	2.79	1.38	1.22
12	364404	227	73	2.78	1.42	1.24
14	3473941	539	113	2.83	1.44	1.23
16	–	1301	211	–	1.45	1.25
18	–	3056	363	–	1.46	1.26
20	–	7276	676	–	1.47	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

Slika 3.12

Na osnovu tabele prikazane na slici 3.12 možemo zaključiti da je h_2 uvek bolje od h_1 . Iz definicije heuristika se jasno vidi da je za bilo koji čvor n $h_2(n) \leq h_1(n)$. Za ovakav slučaj kažemo da h_2 dominira nad h_1 . Diminacija se odražava na efikasnost, jer algoritam sa h_2 heuristikom neće izgenerisati više čvorova od algoritma sa heuristikom h_1 . Dokaz za ovu tvrdnju je jednostavan. Kao što je spomenuto u objašnjenju optimalnosti A^* pretraživanja, algoritam će otvoriti svaki čvor n za koga važi $f(n) < C^*$. Ovo je isto kao da smo rekli da će svaki čvor n za koji važi $h(n) < C^* - g(n)$ biti otvoren. Kako je h_2 manje ili jednako sa h_1 svaki čvor koji se otvori za h_2 biće otvoren i za h_1 , stim što algoritam sa h_1 može otvoriti i neke dodatne čvorove.

3.6.2 Generisanje dopustive heuristike

Heuritičke funkcije možemo formirati na različite načine. Možemo ih dobijati iz pojednostavljenih problema ili iz potproblema. Moguće je implementirati agente koji nauče heuristiku nakon rešavanja velikog broja probelma

Pojednostavljeni problemi

Heuritike h_1 i h_2 za rešavanje problema 8-puzzle su procena putanje koja pločice vodi do cilja, ali su to takođe i tačne putanje za pojednostavljenu verziju problema. Ako bi pravilo bilo da se pločica može pomeriti bilo gde umesto samo u prazno polje tada bi vrednost h_1 označavala tačan broj koraka do rešenja. Slično, ako bi pravilo bilo da se pločica može pomeriti u bilo koji susedni blok, čak i u onaj koji već ima pločicu, tada bi vrednost h_2 predstavljala tačan broj koraka do rešenja.

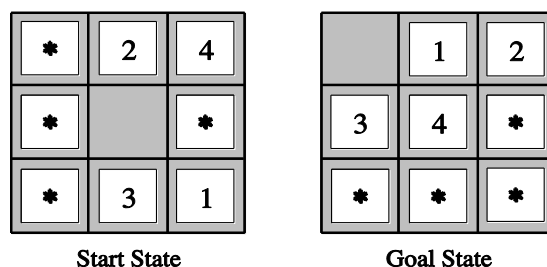
Graf stanja za pojednostavljeni problem je supergraf grafa stanja stvarnog problema, jer uklanjanje restrikcija dodaje nove grane. Kako pojednostavljeni problem samo dodaje grane na prostor stanja stvarnog probelma, svako optimalno rešenje stvarnog probelma je ujedno i rešenje pojednostavljene verzije, s tim što ovde možemo imati i neka bolja rešenja. Zaključak je

da je rešenje pojednostavljenog problema dopustiva heuristika stvarnog problema. Dalje, kako je takva heuristika tačna cena pojednostavljenog problema ona mora zadovoljavati nejednakost trougla pa je i monotona.

Ukoliko imamo više mogućih heuristika za neki problem (h_1, \dots, h_m) , tada možemo reći da je najbolja heuristika $h(n) = \max\{h_1(n), \dots, h_m(n)\}$.

Potproblemi: baza paterna

Dopustiva heuristika se može dobiti i preko potproblema posmatranog problema. Na slici 3.11 prikazan je potproblem 8-puzzle. Samo četiri pločice su obeležene, dok se ostale 4 pomeraju ali njihov raspored nije bitan. Jasno je da je cena optimalnog rešenja potproblema manja od cene kompletnog problema. U nekim slučajevima je čak tačnija od Menhetn razdaljine.



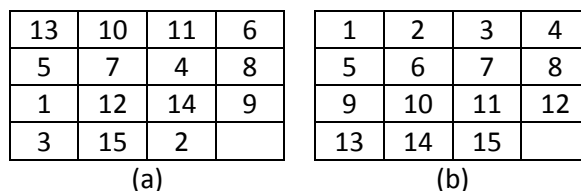
Slika 3.13

Ideja za krairanje baze paterna sastoji se od skladištenja cene rešenja za bilo koju instancu potproblema, u ovom slučaju bilo kog rasporeda četiri pločice i praznog polja. Tada se dopustiva heuristika za svako stanje u stvarnom problemu dobija tako što se u bazi paterna pronade odgovarajuće stanje potproblema i vrati cena njegovog rešenja.

Zadaci

3.1. Navesti jednu netrivialnu dopustivu heuristiku za rešavanje problema 15-puzzle (slika 3.14)?

Koja je vrednost heuristike za stanje prikazano na slici 3.14 (a), ako je ciljno stanje dato na slici 3.14(b)?



Slika 3.14

3.2. Objasniti zašto važe sledeće tvrdnje:

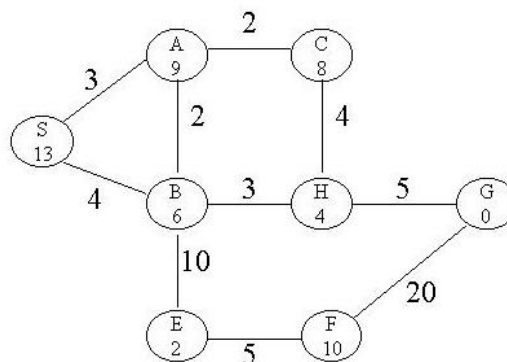
- a) Pretraživanje prvo u širinu je specijalan slučaj pretraživanja vođenog cenom putanje (strategija koja otvara čvor sa najmanjom cenom putanje).
- b) Pretraživanje prvo u dubinu je specijalan slučaj pretraživanja prvo-najbolje (strategija koja otvara čvor sa najmanjom vrednošću heurističke funkcije).
- c) Pretraživanje vođeno cenom putanje je specijalan slučaj A* pretraživanja.

3.3. Posmatrajmo problem pronalaska puta od čvora S do čvora G na grafu koji je prikazan na slici 3.15. Brojevi navedeni na granama predstavljaju cenu koraka za prelaz u susedni čvor, a brojevi u čvorovima vrednost heurističke funkcije za čvor.

Navesti redosled otvaranja čvorova i cene putanje za sledeće strategije pretraživanja:

"Pohlepno" pretraživanje prvo najbolje:

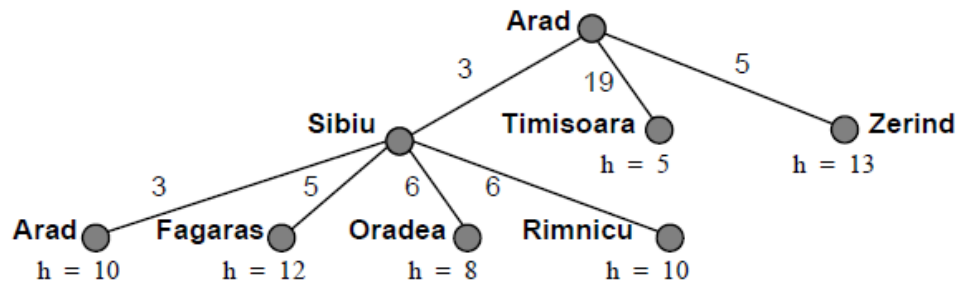
A* pretraživanje:



Slika 3.15

3.4. Na slici 3.16 prikazano je delimično otvoreno stablo pretraživanja za problem pronalaženja puta između gradova u Rumuniji. Na svakoj grani navedena je cena koraka, a za svaki list vrednost heuristike. Koji list će biti sledeći otvoren primenom:

- a) pohlepnog (greedy) algoritma prvo-najbolje?
- b) algoritma vođenog cenom putanje (uniform-cost search)?
- c) A* algoritma?



Slika 3.16

3.5. Posmatramo problem pomeranja k skakača (konja) iz k startnih polja s_1, \dots, s_k u k ciljnih polja g_1, \dots, g_k na beskonačnoj šahovskoj tabli uz pravilo da se dva skakača ne mogu nalaziti u istom polju. Svaka akcija sastoji se od istovremenog pomeranja od 0 do k konja. Treba dostići cilj u najmanjem broju koraka.

- a) koji je maksimalni faktor grananja za ovaj problem.
- b) Pretpostavimo da je h_i dopustiva heuristika za problem pomeranja skakača i na ciljno polje g_i , koja od sledećih heuristika je dopustiva za problem pomeranja k-skakača i koja je najbolja?

(i) $\min\{h_1, \dots, h_k\}$

(ii) $\max\{h_1, \dots, h_k\}$

(iii) $\sum_{i=1}^k h_i$

- c) Koji bi bili odgovori pod b) ako se u svakom potezu pomera samo jedan skakač.

3.6. Pretpostavimo da dva prijatelja žive u različitim gradovima na nekoj mapi (na primer Rumunije prikazanoj na slici 3.1). U svakom potezu pomeramo oba prijatelja istovremeno u jedan od susednih gradova na mapi. vreme potrebno da se iz grada i stigne do susednog grada j jednak je dužini puta $d(i, j)$. Ali, u svakom potezu, prijatelj koji je prvi stigao mora da sačeka dok i drugi ne stigne pre nego što se izvrši sledeći potez. Cilj je da se dva prijatelja sretno što je moguće pre. Dati formalnu definiciju ovog problema, kao problema pretraživanja.

- a) Kako izgleda prostor stanja? (preporuka je da se uvede neka formalna notacije)
- b) Kako izgleda tranzicioni model?
- c) Kako izgleda funkcija cilja?
- d) Koja je cena putanje?
- e) Obeležimo sa $SLD(i,j)$ razdaljinu vazdušnom linijom između bilo koja dva grada i i j . Da li je neka od sledećih dopustiva heuristika za dati problem pretraživanja:
(i) $SLD(i,j)$ (ii) $2 * SLD(i,j)$ (iii) $SLD(i,j)/2$
- f) Da li postoji kompletno povezana mapa za koju ne postoji rešenje?

Poglavlje 4

Ostale tehnike pretraživanja

Strategije pretraživanja opisane u prethodnom poglavlju odnose se na potpuno saglediva deterministička okruženja. U ovom odeljku obrađeni su algoritmi kod kojih nisu u potpunosti zadovoljeni ovi kriterijumi i koji rešavaju probleme bliže realnom svetu.

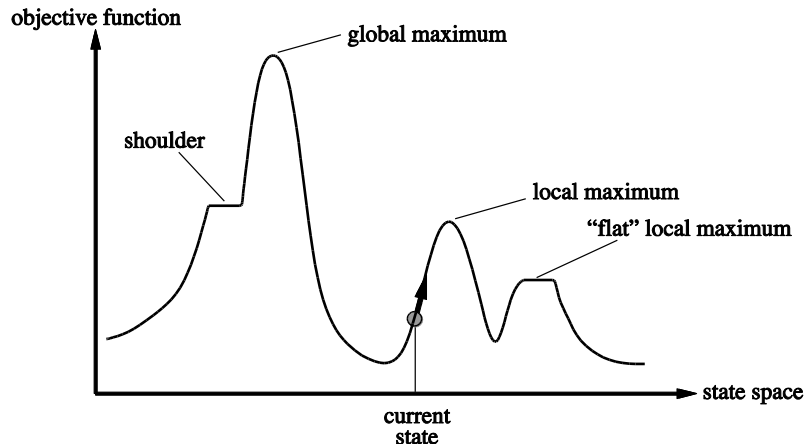
4.1 Algoritmi za lokalno pretraživanje

Umesto sistemskog istraživanja prostora stanja i generisanja putanja od nekog početnog stanja, lokalno pretraživanje predstavlja proces procenjivanja i izmene jednog ili više trenutnih stanja. Ovi algoritmi pogodni su kod onih problema kod kojih nam je samo bitno rešenje (ciljno stanje) ali ne i putanja kojom se dolazi do tog stanja. Primer ovakvog problema je raspored 8 kraljica na šahovskoj tabli. U odeljku 3.2 ovaj problem je formalno opisan tako da se na njega može primeniti neki od standardnih algoritama za neinformisano pretraživanje. Međutim, ono što je bitno kod rešenja ovog problema je konačan raspored kraljica (odnosno ciljno stanje), ali ne i postupak kako smo došli do tog ciljnog stanja odnosno redosled dodavanja kraljica. Ovu osobinu ima veliki spektar problema, kao što su: dizajn integrisanih strujnih kola, dizajn različitih rasporeda (na primer časova u školi), optimizacija telekomunikacione mreže, razni problemi optimizacije...

Osnovni princip lokalnog pretraživanja je da se uzima tekući čvor i pomera se samo do njemu susednog čvora. Ovi algoritmi imaju dve dobre osobine (1) Koriste vrlo malo memorije i (2) često nalaze prihvatljiva rešenja u velikim ili bekonačnim prostorima stanja na koja se ne mogu primeniti sistematični algoritmi za pretraživanje.

Da bismo razumeli lokalno pretraživanje posmatraćemo "pejzaž" (landscape) prostora stanja (slika 4.1). Ovaj pejzaž definisan je preko lokacije definisane stanjem (x osa) i vrednošću heurističke funkcije ili funkcije cilja (y osa). U nekim problemima potrebno je pronaći globalni minimum (na primer ako je u pitanju cena), a u nekim globalni maksimum.

Lokalni algoritmi pretraživanja istražuju ovaj "pejzaž" prostora stanja. Kompletan algoritam za lokalno pretraživanje uvek pronalazi rešenje ako ono postoji, a optimalan algoritam lokalnog pretraživanja uvek pronalazi globalni minimum/maksimum.



Slika 4.1

Algoritam penjanja uzbrdo

Algoritam penjanja uzbrdo prikazan je na slici 4.2. Ovaj algoritam sastoji se od petlje koja se pomera u pravcu većih vrednosti, odnosno uzbrdo. Završava se kada dostigne vrh, odnosno kada ni jedan sused nema veću vrednost.

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

loop do

neighbor \leftarrow a highest-valued successor of *current*

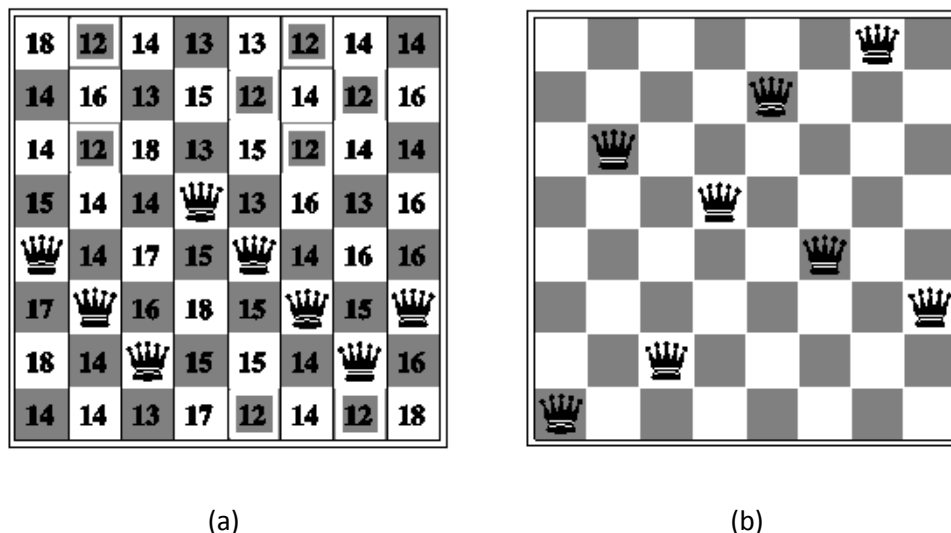
if *neighbor*.VALUE \leq *current*.VALUE **then return** *current*.STATE

current \leftarrow *neighbor*

Slika 4.2

Za ilustraciju algoritma penjanje uzbrdo koristićemo primer rasporeda 8 kraljica na šahovskoj tabli. Lokalno pretraživanje kod ovog problema koristi formulaciju sa kompletnim stanjem (odjeljak 3.2) gde se u svakom stanju svih 8 kraljica nalazi na tabli, u svakoj koloni po jedna. Naredna stanja se dobijaju tako što se jedna kraljica pomeri na neko drugo polje u istoj koloni (što znači da svako stanje ima $8 \cdot 7 = 56$ potomaka). Heuristička funkcija h za ovaj problem predstavlja broj parova kraljica koje se međusobno napadaju, bilo direktno ili indirektno. Globalni minimum za ovu funkciju je 0 i javlja se u idealnom, odnosno ciljnom stanju.

Na slici 4.3a prikazano je stanje sa vrednošću heurističke funkcije $h=17$, a prikazane su i vrednosti heurističke funkcije za naredna stanja. Najbolja vrednost je $h=12$ i algoritam penjanje uzbrdo će slučajnim izborom odabrati jedno od stanja sa $h=12$.



Slika 4.3

Algoritam penjanja uzbrdo se često naziva i gramzivo lokalno pretraživanje jer on uvek "grabi" susedno stanje bez gledanja nekoliko poteza unapred. Ovaj algoritam uvek vrlo brzo napreduje ka cilju. Na primer iz stanja na slici 4.3a ovaj algoritam će u 5 poteza stići u stanje 4.3b za koje je $h=1$, što je vrlo dobro.

Međutim algoritam penjanja uzbrdo se obično zaglavi iz sledećih razloga:

- lokalni maksimum, na slici 4.3 prikazan je lokalni maksimum za problem rasporeda 8 kraljica
- ravnine (shoulder i "flat" local maximum na slici 4.1)

U problemu 8-kraljica algoritam penjanja uzbrdo se zaglavi u 86% slučajeva, a rešava samo 14%. Ukoliko može da pronađe rešenje on ga nađe u 4 koraka. Algoritam se zaglavi ako svi susedi imaju iste vrednosti kao i tekući čvor. Jedan način da se ovo prevaziđe je da se ipak nastavi sa pretraživanjem (pređe u susedni čvor) uz nadu da će se naići na rame (shoulder slika 4.1). Međutim i ovo rešenje se zaglavi u beskonačnoj petlji ako naiđe na ravan lokalni minimum. Rešenje za ovo je da definišemo limit do kog je moguće pomeranje do suseda sa istom vrednošću.

Postoji mnogo varijanti algoritma penjanje uzbrdo. To su stohastičko penjanje uzbrdo (stochastic hill climbing) koje unosi neke verovatnoće u izbor putanja koje "vode" uzbrdo (u zavisnosti od strmosti). Zatim algoritam pravi izbor za penjanje uzbrdo (first-choice hill climbing) koji generiše naslednike u proizvoljnom redosledu dok se ne izgeneriše jedan koji je bolji od tekućeg stanja. Još jedno unapređenje algoritma penjanje uzbrdo je random-restart penjanje uzbrdo koji sprovodi seriju algoritama penjanja uzbrdo iz proizvoljno izgenerisanog početnog stanja.

Lokalno pretraživanje u snopu (local beam search)

Lokalno pretraživanje u snopu prati k stanja umesto jednog (kao kod penjanja uzbrdo). Počinje se sa k proizvoljno igenerisanih stanja i u svakom koraku, svi potomci, svih k stanja se generišu. Ako je bilo koji

od tih potomaka cilj, algoritam se zaustavlja, inače, on selektuje k najboljih potomaka iz cele izgenerisane liste potomaka i nastavlja.

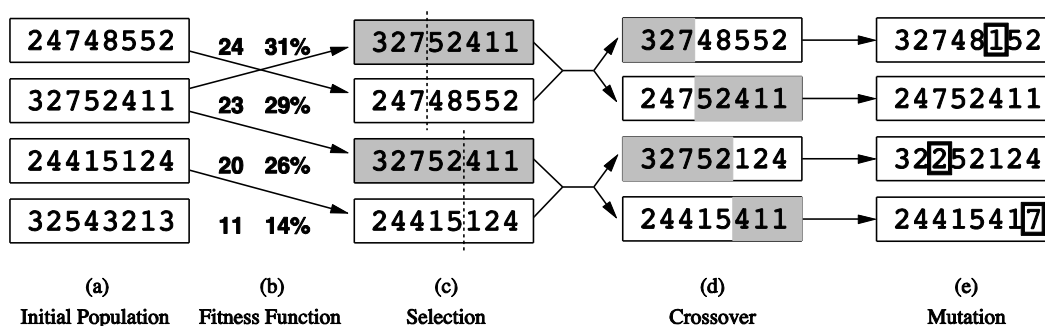
Na prvi pogled ovaj algoritam radi istu stva kao i random-restart osim što se algoritmi ne izvode u sekvenci nego paralelno, međutim ovi algoritmi su potpuno različiti. U random-restart algoritmu svaki proces pretraživanja se odvija nezavisno od ostalih, dok u lokalnom pretraživanju u snopu, korisne informacije se prenose kroz paralelne procese.

Problem sa ovim algoritmom nastaje u slučaju da se izabranih k stanja ne razlikuju mnogo (nisu dovoljno raznovrsna). Tada se dešava da se ta stanja vrlo brzo koncentrišu na malom prostoru i algoritam se svodi na pretraživanje uzbrdo (sa većim memorijskim zahtevima). Varijanta ovog algoritma koja se naziva stohastičko lokalno pretraživanje u snopu ne uzima uvek najboljih k potomaka, već bira k slučajnih potomaka, pri čemu je verovatnoća da neki potomak bude izabran proporcionalna njegovoj vrednosti. Ovaj algoritam podseća na proces prirodne selekcije.

Genetski algoritmi (GA)

Genetski algoritmi su varijanta stohastičkog pretraživanja u snopu u kom se potomci generišu kombinovanjem dva roditelja umesto modifikovanja jednog stanja.

Isto kao kod stohastičkog pretraživanja u snopu, počinje se od k proizvoljno izgenerisanih stanja koji se nazivaju populacija. Svako stanje (ili individua) predstavljeno je stringom (korišćenjem nekog konačnog alfabeta), najčešće nizom nula i jedinica. Na primer, u problemu 8 kraljica stanje se može predstaviti nizom od osam brojeva koji predstavljaju poziciju kraljice u svakoj koloni. Na slici 4.5 prikazano je četiri stanja u problemu 8 kraljica. Produkcija sledeće generacije prikazana je na slici 4.5 (b-e). U (b) svako stanje se ocenjuje na osnovu „fitness“ funkcije koja vraća veće vrednosti za bolja stanja. Na primer za problem 8-kraljica to može biti broj parova kraljica koje se ne napdaju (za rešenje je to 28). Na slici (4.5b) prvi broj predstavlja vrednost fitness funkcije za stanje, a pored tog broja prikazana je verovatnoća da stanje bude izabrano u reprodukciji. Ta verovatnoća je proporcijalna vrednošću fitness funkcije.



Slika 4.5

U 4.5 (c) selektuje se četiri stanja na osnovu prikazanih verovatnoća. Treba primetiti da je jedno stanje selektovano dva puta, a da je jedno stanje izostavljeno. Na ovaj način izabrana su dva para koje će se ukrstiti. Za svaki od dva para proizvoljno se bira pozicija (crossover) u stringu. Na slici 4.5 za prvi par ta

pozicija je posle treće cifre, a za drugi posle 5. U (d) se kreiraju potomci tako što se roditeljima zamene stringovi posle izabrane pozicije. Ako se roditelji mnogo razlikuju, njihov potomak će se takođe mnogo razlikovati od oba roditelja, tako da je obično slučaj da ovaj proces značajno menja stanja u početku, kada je populacija raznovrsna, a da je kasnije ta promena manja, kada su stanja sličnija. Konačno u (e) vrši se proizvoljna mutacija sa malom nezavisnom verovatnoćom. Po jedna cifra je mutirana u prvom, trećem i četvrtom stringu. U problemu 8 kraljica ova mutacija predstavlja proces uzimanja proizvoljne kraljice i pomeranja u proizvoljno polje u istoj koloni.

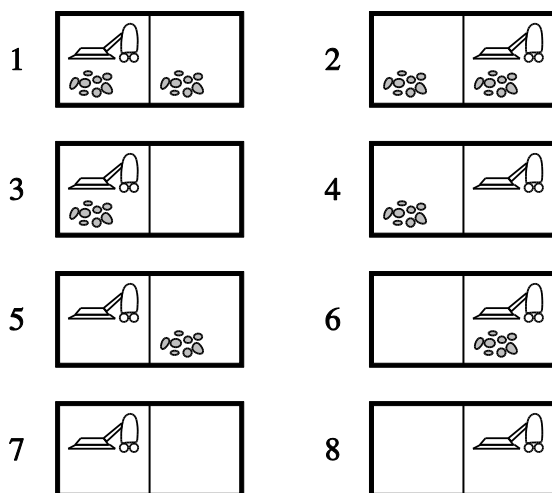
Slično kao stohastičko pretraživanje u snopu, genetski algoritmi kombinuju tehniku penjanja uzbrdo sa proizvoljnim istraživanjem prostora stanja i razmenom informacija između paralelnih procesa pretraživanja. Osnovna prednost genetskih algoritama je u procesu ukrštanja roditelja. Na primer, ako napravimo početni blok 2, 4, 6 u kojima se kraljice ne napadaju, taj blok predstavlja koristan blok koji se može kombinovati sa ostalima do rešenja.

U praksi genetski algoritmi imaju velikog uticaja u domenu optimizacije, ali ostaje još posla u identifikaciji uslova pod kojim genetski algoritmi daju dobre rezultate.

4.2 Pretraživanje sa nedeterminističkim akcijama

Ako posmatramo pretraživanje u nedeterminističkom okruženju u kom agent ne može sa sigurnošću da zna u koje stanje će ga odvesti akcija, tada je rešenje problema ne skup akcija, već plan za nepredvidive situacije (contingency plan) ili strategija. Ovakav plan govori šta treba uraditi u zavisnosti od zapažanja koja nam govore u kom stanju se nalazimo. Ovo vrsta pretraživanja u nastavku je ilustrovana na primeru neispravnog usisivača.

Problem koji smo analizirali u odeljku 3.2 sastojao se od okruženja koje ima dve prostorije i usisivač. Na slici 4.6 prikazana su sva moguća stanja u kojima se može nalaziti agent-usisivač. Cilj je da se usisaju obe prostorije, odnosno da se dođe u stanje 7 ili 8.



Slika 4.6

Sada pretpostavimo da su akcije usisivača nedeterminističke i da akcija usisaj (Suck) radi na sledeći način:

- Kada se pozove za prljavo polje usisa se to polje, a ponekad se usisa i susedno polje.
- Kada se pozove u čistom polju usisivač ponekad prospe prašinu na pod (☺).

Tranzicioni model za ovakav problem ne može se postaviti kao u slučaju ispravnog usisivača. Funkcija Result vraća skup stanja u kojima se agent može naći ako izvrši određenu akciju. Na primer, akcija Suck u stanju 1 može kao rezultat imati stanja {5,7}.

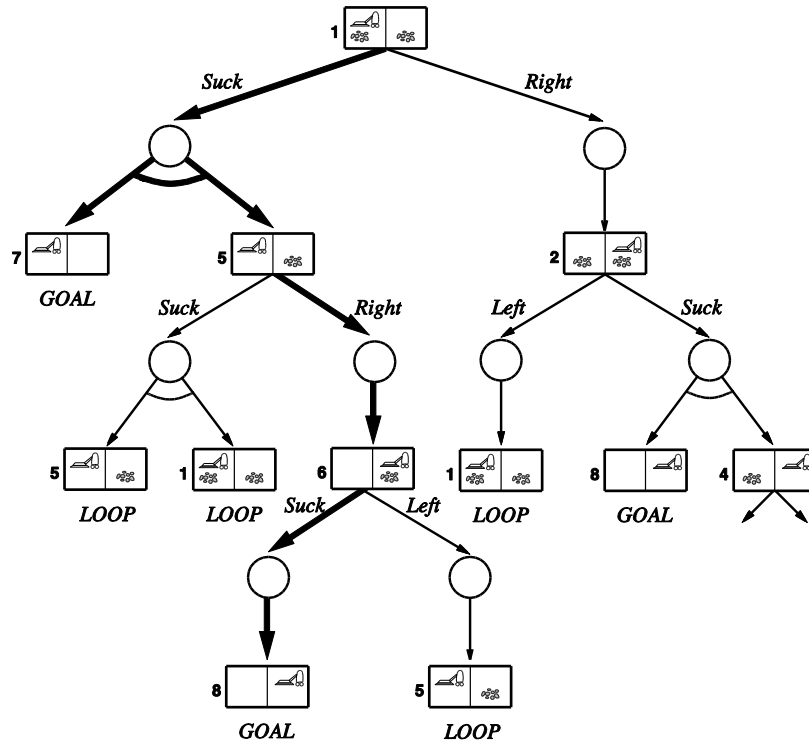
Takođe rešenje za problem nije samo niz akcija već plan za nepredvidive situacije koji (iz stanja 1) izgleda ovako:

[Suck, if State=5 then [Right, Suck] else []].

Znači, za nedeterministička okruženja rešenja mogu sadržati if-else izraze, što znači da se ova rešenja mogu predstaviti kao stabla umesto sekvence (akcija). Mnogi problemi realnog sveta imaju nedeterministički karakter.

AND-OR stabla pretraživanja

Stabla pretraživanja za nedeterminističke akcije nazivaju se AND-OR stabla. OR čvorovi su oni u kojima agent odlučuje šta će da uradi, a AND čvorovi su oni koji predstavljaju „odluku“ okruženja koja akcija će ustvari da se izvrši. Jedno AND-OR stablo prikazano je na slici 4.7. Rešenje ovakvog stabla pretraživanje je njegovo podstablo čiji svaki list predstavlja ciljno stanje, sadrži jednu granu iz svakog OR čvora i sve grane koje izlaze iz AND čvorova. Na slici 4.7 rešenje je prikazano boldovanim granama.



Slika 4.7

Pokušaj, pokušaj ponovo

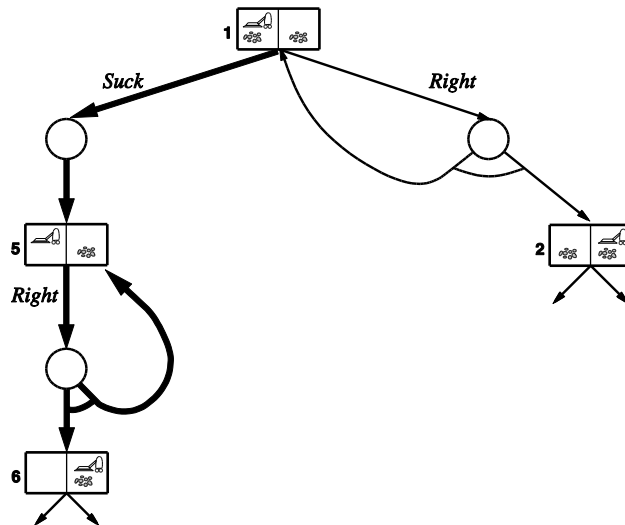
Ako sada posmatramo usisivač koji radi ispravno, s tim što se on kreće u „klizavom“ okruženju, odnosno njegove akcije kretanja mogu da se ne izvrše pravilno. Na primer, iz stanja 1, izvršavanjem akcije Right on može doći u stanje 1 ili u stanje 2 (slika 4.6). Na slici 4.8 prikazan je deo grafa pretraživanja. Jasno je da graf verzija pretraživanja ne bi dovela do rešenja, jer ne postoji aciklično rešenje za ovaj problem. Međutim, postoji ciklično rešenje koje se sastoji od toga da usisivač pokušava da izvrši akciju Right sve dok ga ona ne odvede u željeno polje.

Rešenje za ovaj problem se može zapisati uvođenjem promenljive L_1 koja označava neki deo plana koji se ponavlja, tako da se onda na ponovljenom mestu piše samo naziv promenljive, a ne i ceo plan:

[Suck, L_1 :Right, if State=5 then L_1 else Suck].

(mada bi za deo sa petljom bolja sintaksa bila „while State=5 do Right“).

Uopšteno govoreći, plan ili strategija koja sadrži petlje jeste rešenje problema ukoliko je svaki list ciljno stanje i ako se do svakog lista može stići iz bilo kog čvora u stablu. U takvoj situaciji agent će sigurno doći do cilja ako pretpostavimo da će se svaka nedeterministička akcija jednom izvršiti.



Slika 4.8

4.3 Pretraživanje u okruženju koje se ne može potpuno sagledati

U ovom odeljku obrađen je slučaj okruženja u kom agent ne može sa sigurnošću da zna u kom stanju se nalazi posle izvršenja neke akcije, čak iako je okruženje determinističko. Ono što agent ima je njegovo mišljenje odnosno verovanje da se nalazi u određenom stanju (belief state).

Pretraživanje bez zapažanja

Prvi slučaj je agent koji nema senzora i koji ne dobija nikakve informacije iz okruženja. Na prvi pogled se može učiniti da ovakav agent ne može da reši ni jedan problem, međutim postoje problemi u kojima se on vrlo dobro "snalazi" (primer: doktor će prepisati antibiotik pre nego što izvrši seriju skupih testova i na koje bi pacijent trebalo da čeka jako dugo što bi dovelo do pogoršanja njegovog stanja).

Posmatrajmo slučaj agenta usisivača koji ne zna u kojoj lokaciji se nalazi i ne zna da li je ta lokacija prljava (ali zna da njegovo okruženje ima dve prostorije). Bilo koje stanje na slici 4.6 može biti početno stanje. Ukoliko u bilo kom od stanja sa slike 4.6 agent izvrši akciju *Right* on će se naći u nekom od stanja {2,4,6,8}. Ako posle toga izvrši akciju *Suck*, naći će se u jednom od stanja {4,8}. Konačno, sekvenca akcija *Right*, *Suck*, *Left*, *Suck* će sigurno agenta dovesti u ciljno stanje 7, bez obzira koje je početno stanje.

Kod agenata koji nemaju senzora, umesto grafa realnih stanja, pretražuje se graf "verovatnih" stanja (belief state). U takvom grafu, iz tačke gledišta agenta on se nalazi u potpuno sagledivom okruženju jer on uvek zna koja su mu verovatna stanja.

Kako se formira kompletna definicija ovakvih problema? Za neki problem P definišu se sledeći elementi:

- Verovatna stanja: Prostor svih mogućih verovatnih stanja (jedan čvor sadrži skup stanja za koje agent "veruje" da se u njima trenutno nalazi). Ako problem P ima N mogućih stanja onda je broj verovatnih stanja 2^N (broj mogućih podkupova skupa sa N elemenata)

- Početno stanje: U najopštijem slučaju to je skup svih mogućih stanja problema P.

- Akcije iz nekog stanja: Pretpostavimo da je agent u verovatnom stanju $b=\{s_1, s_2\}$ i da je $ACTIONS_p(s_1) \neq ACTIONS_p(s_2)$, gde $ACTIONS_p(s)$ predstavlja akcije dostupne u stanju s prema definiciji problema. Ako pretpostavimo da akcije koje nisu dozvoljene nemaju nikakvog efekta onda važi:

$$ACTIONS(b) = \bigcup_{s \in b} ACTIONS_p(s)$$

Sa druge strane ako akcije koje nisu dozvoljene imaju loš efekat ili mogu agenta odvesti u neko neželjeno stanje, onda bi skup dozvoljenih akcija bio presek svih dozvoljenih akcija u stanjima koje pripadaju skupu verovatnih stanja.

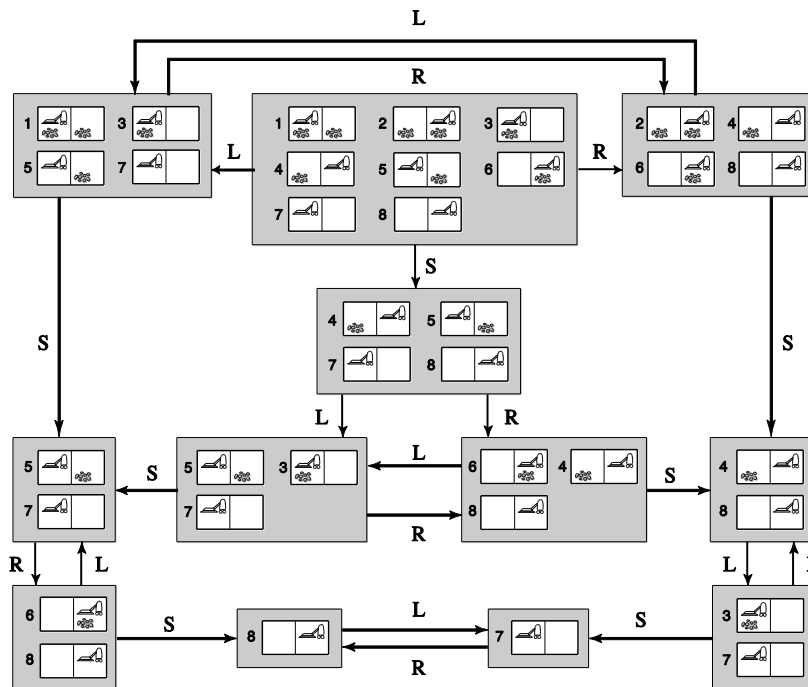
- Tranzicioni model: Agent ne zna u kom je tačno stanju iz skupa verovatnih stanja, što znači da nakon izvršavanja akcije on može biti u bilo kom stanju dok se može stići izvršavanjem određene akcije u jednom od verovatnih stanja. Za determinističke akcije tranzicioni model se može zapisati na sledeći način:

$$b' = RESULT(b, a) = \{s' : s' \in RESULT_p(s, a) \text{ and } s \in b\}$$

- Funkcija cilja: Verovatno stanje je ciljno stanje ako su sva stanja u skupu verovatnih stanja ciljna stanja. Može se desiti da i pre toga agent dođe u ciljno stanje, a da toga nije svestan.

- Cena putanje: Ukoliko se cena iste akcije može razlikovati u različitim stanjima tada u prostoru verovatnih stanja ta cena može biti jedna od mogućih cena akcije. Ako pretpostavimo da je cena akcije uvek ista onda se cene putanja mogu preslikati iz inicijalne definicije problema P.

Graf stanja ovako opisanog probčema dat je na slici 4.9.



Slika 4.9. Graf verovatnih stanja za usisivač bez senzora

Kada se problem za agenta bez senzora opiše na ovakav način na njega se može primeniti jedan od standardnih algoritama za pretraživanje prostora stanja. Međutim, većina realnih problema se ne može rešiti na ovaj način. Prvi problem je veliki faktor grananja, dok je drugi, još veći problem veličina verovatnih stanja (na primer za svet usisivača veličine 10X10 inicijalno verovatno stanje sadrži $100 \cdot 2^{100}$ što je oko 10^{32} realnih stanja).

Jedno rešenje za ovakve probleme je da ne koristimo algoritme pretraživanja i da verovatna stanja ne posmatramo kao crne kutije, već da analiziramo moguća realna stanja i da rešavamo jedno po jedno realno stanje. Na primer, za problem usisivača posmatramo inicijalno verovatno stanje $\{1,2,3,4,5,6,7,8\}$ i tražimo rešenje za realno stanje 1. kad ga pronađemo, proveravamo da li to rešenje važi i za ostala stanja. Ukoliko ne, vraćamo se na stanje 1 i tražimo sledeće rešenje.

Pretraživanje sa parcijalnim zapažanjem

Primer okruženja koje se može parcijalno sagledati je svet usisivača sa lokalnim senzorom u kom usisivač ima senzor koji mu govori u kom polju se nalazi i senzor koji mu govori da li je to polje zaprljano, ali nema senzor koji mu govori da li je susedno polje uprljano. Za formalnu specifikaciju ovakvih problema potrebna nam je funkcija $PERCEPT(s)$ koja vraća sva zapažanja agenta u stanju s . Na primer za usisivač sa lokalnim senzorom vrednost ove funkcije u stanju 1 (slika 4.6) je $[A, Dirty]$. Za okruženja koja se mogu potpuno sagledati važi $PERCEPT(s)=s$, a za okruženja koje se uopšte ne može sagledati važi $PERCEPT(s)=null$.

Ukoliko imamo parcijalna zapažanja može se desiti da više stanja proizvede ista zapažanja. Na primer, zapažanja [A, Dirty] će proizvesti stanja {1,3} odnosno ako imamo ovakvo zapažanje u početnom stanju tada takav skup predstavlja početno verovatno stanje. Skup mogućih akcija, cena putanje i funkcija cilja se definišu na isti način kao kod agenta bez senzora. Međutim, u slučaju parcijalnih zapažanja tranzicioni model je nešto složeniji i definiše se preko sledeća tri koraka:

- Prvi korak je predikcija koja govori koja sve verovatna stanja možemo dobiti primenom akcije a u verovatnom stanju b , $\hat{b} = PREDICT(b, a)$
- Drugi korak je predikcija zapažanja koja predviđa sva moguća zapažanja u nekom verovatnom stanju b . $POSSIBLE - PERCEPTS(\hat{b}) = \{o: o = PERCEPT(s) \text{ and } s \in \hat{b}\}$
- Treći korak je ažuriranje u kom se za svako zapažanje određuje verovatno stanje koje može proizvesti to zapažanje. $b_o = UPDATE(\hat{b}, o) = \{s: o = PERCEPT(s) \text{ and } s \in \hat{b}\}$.

Kada se povežu ova tri koraka dobijamo sledeću definiciju tranzicionog modela:

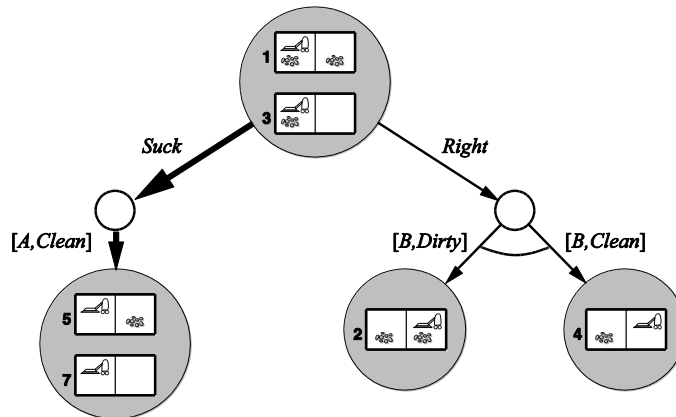
$$RESULT(b, a) = \{b_o: b_o = UPDATE(PREDICT(b, a), o) \text{ and } o \in POSSIBLE - PERCEPTS(PREDICT(b, a))\}$$

Na ovakve probleme mogu se primeniti AND-OR stabla pretraživanja opisana u odeljku 4.2. Na slici 4.10 prikazano je AND-OR stablo za problem usisivača sa lokalnim senzorom za inicijalno zapažanje [A, Dirty]. Rešenje ovakvog problema je takođe plan za nepredviđene situacije:

[Suck, Right, if Bstate={6} then Suck else []].

Implementacija agenta za rešavanje problema u parcijalno sagledivom okruženju je slična implementaciji agenta za rešavanje problema pretraživanjem (odeljak 3). Agent formuliše problem i sprovodi neki od algoritama pretraživanja (na primer Graf search na AND-OR stablu) koji mu vraća rešenje problema. Razlika je u tome što kod ovog agenta rešenje nije sekvenca akcija već plan za nepredviđene situacije koji se sastoji od if-else izraza. Agent mora da testira uslove navedene u if-else i da na osnovu toga izvršava određene akcije. Druga značajna stvar kod agenta koji izvršava akcije u parcijalno sagledivom okruženju je što on mora da ažurira svoja verovatna stanja u zavisnosti od zapažanja. Odnosno novo verovatno stanje b' za prethodno verovatno stanje b , nakon izvršene akcija a i na osnovu zapažanja o se dobija na sledeći način:

$$b' = UPDATE(PREDICT(b, a), o).$$



Slika 4.10

Okruženja koja se samo parcijalno mogu sagledati obuhvataju veliki broj problema iz realnog života i proces manipulasinja verovatnim stanjima predstavlja osnovu funkciju mnogih inteligentnih sistema.

Robot (strana 145/146 u AIMA)

4.4 Online pretraživanje u napoznatom okruženju

Do sada smo razmatrali agente koji koriste offline pretraživanje. Oni računaju rešenje pre nego što zakorače u stvaran svet. Pored offline pretraživanja postoji i online pretraživanje koje kombinuje pretraživanje sa konkretnim akcijama. Ova vrsta pretraživanja se naročito koristi u dinamičnim i poludinamičnim okruženjima gde agent dobija penale ako računa suviše dugo. Takođe, online pretraživanje je korisno i kod nedeterminističkih okruženja jer omogućava agentu da se skoncentriše na stanja koja su se zaista desila, a da ne gubi vreme na ona koja će se možda desiti.

Online pretraživanje je neophodno u nepoznatim okruženjima, u kojima agent ne zna koja sve stanja postoje i koji su rezultati njegovih akcija. U ovakvim okruženjima agent se susreće sa problemom istraživanja okruženja (exploration problem) u kojima koristi svoje akcije kao eksperimente da bi dobio informacije o izgledu okruženja. Primer agenat u nepoznatom okruženju je robot koji je postavljen u novu zgradu i mora da je istraži kako bi pronašao put od A do B.

Formulacija problema za online pretraživanje u determinističkom i potpuno sagledivom okruženju sastoji se od tri parametra:

- ACTIONS(s) – skup dozvoljenih akcija u određenom stanju
- cena koraka $c(s,a,s')$ koja se može izračunati tek kad agent zaista dođe u stanje s' .
- GOAL-TEST(s).

U nekim slučajevima agent može da ima i vrednost dopustive heuristike $h(s)$ koja procenjuje udaljenost trenutnog stanja od ciljnog stanja.

Agent ne zna unapred $RESULT(s,a)$ već će ga saznati onog trenutka kada se nađe u s i izvrši a . On ništa ne računa unapred, već problem rešava izvršavanjem akcija. Kada izvrši akciju on dobije zapažanje koje mu govori u kom stanju se nalazi. Na taj način on formira mapu koja mu pomaže da odluči gde će ići dalje. Zbog ovih osobina online pretraživanje se veoma razlikuje od na primer A^* pretraživanja jer ono ne može da prvo otvori jedan čvor, pa zatim neki udaljeni čvor, jer su njegovi potezi stvarne akcije, a ne simulacije akcija. Algoritam koji bi odgovarao ovakvom tipu pretraživanja je pretraživanje prvo u dubinu. Takođe i razne varijante lokalnih pretraživanja (penjanje uzbrdo) se mogu u izmenjenom obliku primeniti na ovakve probleme. Na primer, algoritam penjanja uzbrdo sa memorijom.

Zadaci

4.1. Koje algoritme dobijamo u sledećim specijalnim slučajevima:

a) lokalno pretraživanje u snopu za $k=1$

b) lokalno pretraživanje u snopu za jedno početno stanje i bez ograničenja na broj potomaka u narednim koracima

4.2. Posmatramo problem usisivača bez senzora. Pretpostavimo da agent zna $h^*(s)$ -stvarnu cenu optimalnog rešenja iz stanja s u potpuno sagledivom okruženju. Naći jednu optimalnu heuristiku za verovatno stanje b ($h(b)$) izraženu preko $h^*(s)$ za $s \in b$. Koje su vrednosti heurističke funkcije za verovatna stanja sa slike 4.9? Koji je redosled otvaranja čvorova po A^* pretraživanju korišćenjem date heurističke funkcije?

4.3. Posmatramo svet usisivača sa nedeterminističkim akcijama (kao u odeljku 4.2) koji nema senzora. Nacrtati prostor verovatnih stanja dostupnog iz početnog verovatnog stanja $\{1,3,5,7\}$. Da li ovaj problem ima rešenje?

Poglavlje 5

Pretraživanje u igrama

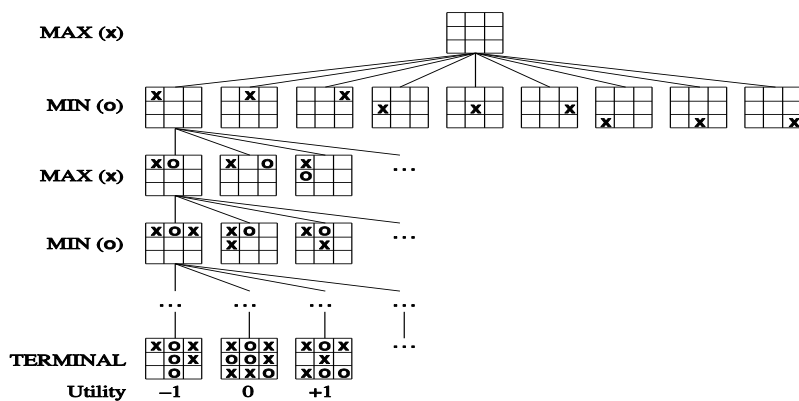
Posmatramo okruženje sa više agenata u kom agenti moraju da razmatraju akcije drugih agenata i uticaj tih akcija. U ovom poglavlju koncentrišemo se na kompetitivno ili takmičarsko okruženje u kojima su ciljevi agenta u konfliktu - igre (adversarial search).

Najčešća vrsta igara u VI su determinističke igre u kojima igrači (najčešće imamo dva igrača) igraju naizmenično i ishod je takav da je jedan od igrača pobedio, drugi izgubio ili je nerešeno. Ovakve igre imaju stanja koja se jednostavno predstavljaju u programima, i agenti obično imaju mali skup mogućih akcija.

Posmatramo igre sa dva igrača koje ćemo zvati MIN i MAX. MAX je prvi na potezu, zatim MIN i tako naizmenično dok se igra ne završi. Igra se može formalno opisati kao problem pretraživanja preko sledećih elemenata:

- S_0 - početno stanje
- Player(s) - koji igrač je na potezu u stanju s
- Actions(s) - skup legalnih poteza u stanju s
- Result(s,a) - tranzicioni model koji definiše rezultat poteza
- Utility(s,p) - funkcija cilja, daje konačan rezultat u završnom stanju s za igrača p, na primer u šahu rezultat ove funkcije može biti pobeda, poraz ili nerešeno, sa vrednostima +1,0,1/2

Početno stanje, akcije i tranzicioni model definišu stablo pretraživanja u igrama u kome su čvorovi stanja u igri a grane su potezi. Na slici 5.1 prikazano je stablo pretraživanja za igru iks-oks. Iz početnog stanja MAX ima 9 mogućih poteza. Na svakom nivou jedan od igrača MIN i MAX je na potezu i smenjuju se naizmenično. Igra se završava kada jedan od igrača napravi tri u nizu ili su sva polja popunjena. Za svaki list se vezuje vrednost funkcije cilja sa tačke gledišta MAX-a, veće vrednosti su bolje za MAX-a, a manje za MIN-a, po čemu su igrači i dobili ime.

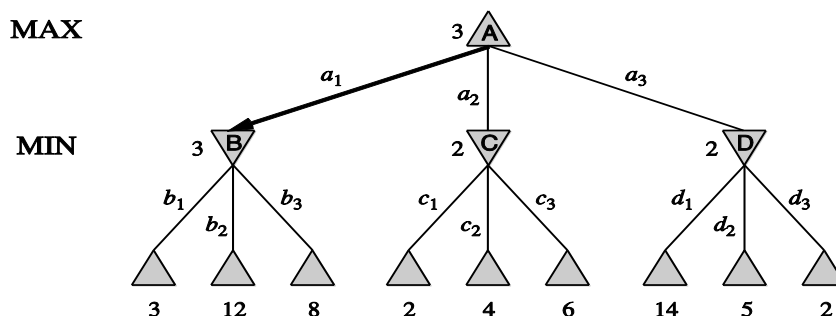


Slika 5.1

5.1 Optimalno odlučivanje u igrama

U klasičnim tehnikama pretraživanja, optimalno rešenje je skup akcija koje vode do cilja, završno stanje koje znači pobjedu. U igrama imamo drugog igrača (MIN) čije akcije moramo da razmotrimo. Zbog toga MAX igrač mora da nađe strategiju koja određuje njegov potez u početnom stanju, zatim odgovor na svaki mogući potez MIN igrača i tako dalje. Pokazaćemo kako da se pronađe optimalna strategija.

Na slici 5.2 prikazano je stablo pretraživanja u trivijalnoj igri.



Slika 5.2

Strategija u ovoj igri je definisana na sledeći način:

$$MINIMAX(s) = \begin{cases} UTILITY(s) & \text{ako je } s \text{ završno stanje} \\ \max_{a \in Actions(s)} MINIMAX(Result(s, a)) & \text{ako je } Player(s) = MAX \\ \min_{a \in Actions(s)} MINIMAX(Result(s, a)) & \text{ako je } Player(s) = MIN \end{cases}$$

Stablo na slici 5.2 se formira tako što listovi dobiju vrednost funkcije cilja. MIN čvorovi uzimaju najmanju vrednost svojih potomaka, a MAX čvorovi najveću vrednost svojih potomaka. Prvi MIN čvor ima tri potomka, sa vrednostima 3, 12 i 8 tako da je njegova vrednost MINIMAX-a 3. Na isti način se formiraju vrednosti ostalih MIN čvorova. Koren je MAX čvor koji ima tri potomka čije su vrednosti 3, 2 i 2, a njihova maksimalna vrednost je 3. U korenu identifikujemo odluku minimax algoritma, a to je akcija a_1 koja nas vodi u stanje koje ima maksimalnu vrednost.

MINIMAX algoritam u pseudo kodu prikazan je na slici 5.3. Ovaj algoritam koristi rekursiju za kreiranje MINIMAX stabla pretraživanja. Funkcija se poziva rekursivno sve do listova, gde se izračuna vrednost funkcije cilja, a zatim se te vrednosti na izlasku iz rekursije propagiraju dalje u pliće čvorove.

Kakve su osobine MINIMAX algoritma? On je sličan pretraživanju prvo u dubinu. Obeležićemo vrednosti na sledeći način:

- m - dubina stabla pretraživanja MINIMAX algoritmom,
- b - broj legalnih poteza u nekom stanju.

Vremenska složenost algoritma je $O(b^m)$, a prostorna složenost $O(bm)$ za verziju algoritma koja generiše sve akcije odjednom, odnosno $O(m)$ za verziju koja generiše jednu po jednu akciju. Za realne igre, vremenska složenost je apsolutno nepraktična, ali ovaj algoritam je osnova za matematičku analizu igara i za druge praktičnije algoritme.

```

function MINIMAX-DECISION(state) returns an action
  return  $\arg \max_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$ 



---


function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$ 
  return v



---


function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow \infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$ 
  return v

```

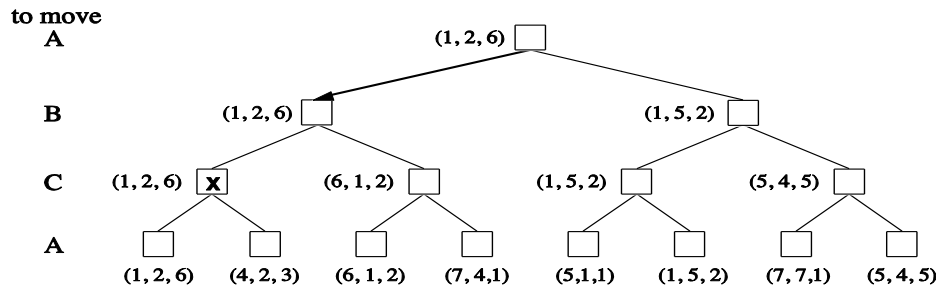
Slika 5.3

5.1.1 Optimalno odlučivanje u igrama sa više igrača

Mnoge popularne igre imaju više od dva igrača i postavlja se pitanje kako se MINIMAX algoritam može proširiti za ovakve igre.

Kod igara sa dva igrača imamo samo jednu vrednost (jer je druga bila suprotna), dok kod igara sa više igrača moramo uvesti vektor. Na primer, u igrama sa tri igrača A, B i C imaćemo vektor $\langle v_A, v_B, v_C \rangle$ koji će se vezivati za svaki čvor. U završnom stanju vektor sadrži vrednosti funkcije cilja sa tačke gledišta svakog od tri igrača.

Na slici 5.4 prikazano je stablo pretraživanja za igru sa tri igrača. U čvoru X je na potezu igrač C i on bira vrednost koja njega vodi u bolji rezultat, a to je 6, umesto 3 (posmatramo treću poziciju u potomku).

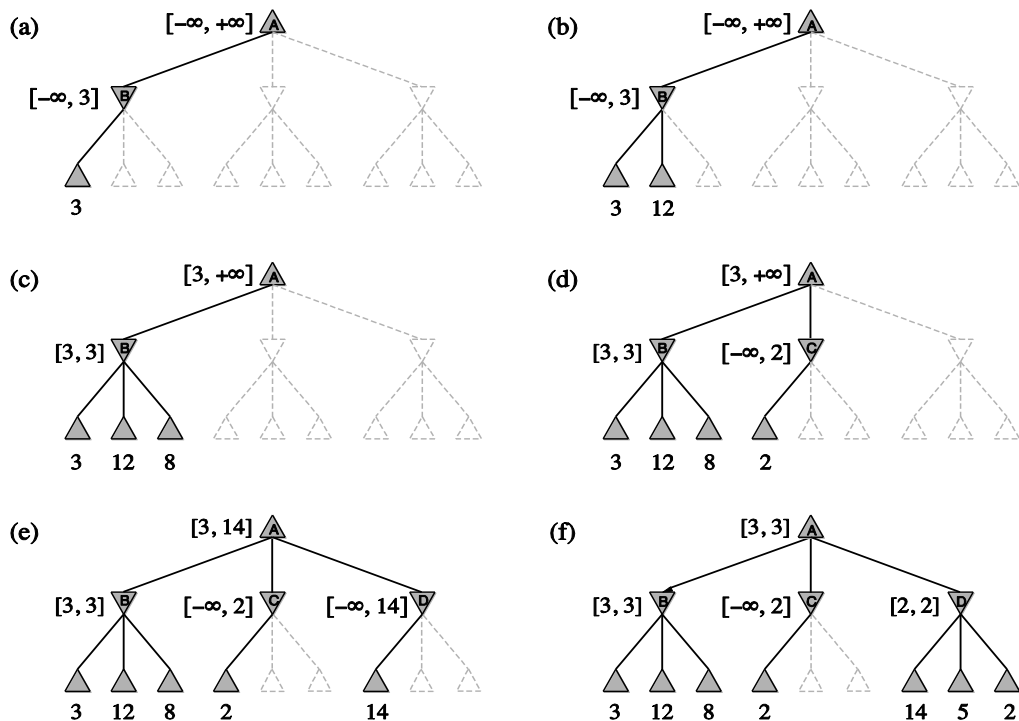


Slika 5.4

5.2 Alfa-beta odsecanje

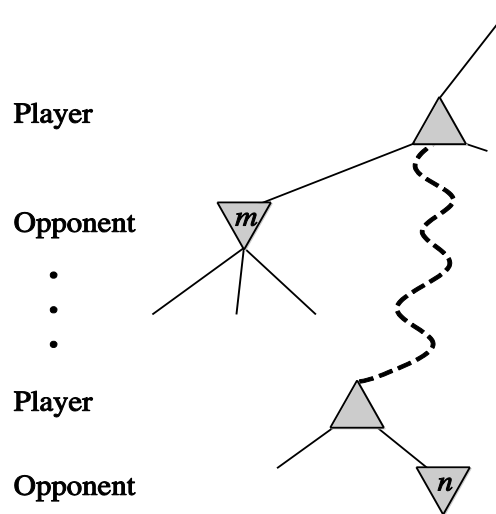
Problem sa MINIMAX algoritmom je što je broj stanja u igri koje treba ispitati eksponencijalna vrednost dubine stabla. Međutim, skoro polovina stanja se može "odseći" iz stabla jer ne utiču na krajnji rezultat. Tehnika za ovaj postupak zove se alfa-beta odsecanje. Kada se primeni na standardno MINIMAX stablo pretraživanja vraća isti potez kao rezultat, ali odseca veliki broj grana koje ne utiču na krajnju odluku.

Na slici 5.5 prikazano je stablo sa slike 5.2 na koje je primenjeno alfa-beta odsecanje. Ideja je da možemo da izračunamo konačan rezultat bez otvaranja svih listova. Pošto je rezultat u čvoru B jednak 3, a jedan od potomaka čvora C je 2, minimum potomaka od C je sigurno manji ili jednak 2, a to je sigurno manje od 3 što znači da će čvor C sigurno imati manju vrednost od čvora B i neće biti izabran kao maksimalna vrednost. To znači, da ostale potomke čvora C ne moramo da otvaramo.



Slika 5.4

Alfa-beta algoritam se može primeniti na bilo koju dubinu i njime se najčešće odsecaju cela podstabla, a ne samo listovi. Generalni princip je sledeći: posmatramo neki čvor n u stablu pretraživanja (Slika 5.5) tako da igrač ima opciju da izabere taj čvor. Ako igrač ima neku bolju opciju m bilo u čvoru roditelju od n bilo u nekom plicem čvoru tada se čvor n nikada neće dostići u igri. Znači da kada saznamo dovoljno o n (otvaranjem određenog broja potomaka) da ovo zaključimo možemo ga odseći.



Slika 5.5

Alfa-beta odsecanje je dobilo naziv na osnovu dva parametra koji se formiraju tokom pretraživanja

- alfa - najbolja vrednost za MAX-a (najveća vrednost),
- beta - najbolja vrednost za MIN-a (najmanja vrednost).

Alfa-beta pretraživanje ažurira vrednosti parametara alfa i beta u toku pretraživanja i odseca nepotrebne grane iz čvora sve dok se zna da je vrednost tekućeg čvora gora od trenutnih vrednosti alfa i beta za MAX i MIN igrača redom. Na slici 5.6 prikazan je MINIMAX algoritam sa alfa-beta odsecanjem u pseudo kodu.

Broj odsečenih grana u MINIMAX algoritmu sa alfa-beta odsecanjem zavisi dosta od redosleda otvaranja čvorova. Na primer, ako posmatramo sliku 5.4 (f) vidmo da su i neki potomci čvora D mogli biti odsečeni samo da je redosled čvorova bio drugačiji.

```

function ALPHA-BETA-SEARCH(state) returns an action
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
  return the action in ACTIONS(state) with value v

```

```

function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \geq \beta$  then return v
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return v

```

```

function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for each a in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \leq \alpha$  then return v
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return v

```

Slika 5.6

5.3 MINIMAX do određene dubine

MINIMAX algoritam generiše celo stablo pretraživanja u igri, dok alfa-beta algoritam omogućava da se odseče veliki deo tog stabla. Međutim, i alfa-beta verzija algoritma mora da pretraži do završnog stanja makar za deo stabla. Ova dubina je obično nepraktična, jer potez treba da se odigra u razumnom vremenu. Umesto uopštenog MINIMAX algoritma u praksi se najčešće primenjuje MINIMAX do neke zadate dubine na kojoj se izračunava heuristička funkcija evaluacije za listove koji nisu završna stanja.

$$H - \text{MINIMAX}(s, d) =$$

$$\begin{cases} \text{EVAL}(s) & \text{ako je } s \text{ kraj igre ili smo} \\ & \text{dostigli zadatu dubinu} \\ \max_{a \in \text{Actions}(s)} H - \text{MINIMAX}(\text{Result}(s, a), d + 1) & \text{ako je Player}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} H - \text{MINIMAX}(\text{Result}(s, a), d + 1) & \text{ako je Player}(s) = \text{MIN} \end{cases}$$

5.3.1 Funkcije evaluacije u igrama

Funkcija evaluacije daje procenu očekivanog rezultata u igri iz trenutne pozicije, slično kao kod klasičnog pretraživanja gde je procenjivana udaljenost od cilja. Očigledno je da uspešnost programa koji igra u velikoj meri zavisi od kvaliteta funkcije evaluacije. Pitanje je kako napraviti dobru funkciju evaluacije.

Prvo, funkcija evaluacije treba da uredi listove stabla pretraživanja na isti način kao funkcija cilja, stanja koja vode u pobjedu moraju biti procenjena boljom vrednošću nego ona koja vode u nerešeno, a ova boljom od onih koja vode u poraz. Drugo, izračunavanje funkcije ne treba da bude predugo.

Mnoge funkcije evaluacije se formiraju tako što se daju numeričke vrednosti nekim elementima igre. Na primer u šahu, broji se broj topova, konja, lovaca i belih i crnih,... Zatim se svakoj od ovih figura može pridružiti i neki koeficijent (materijalna vrednost), tako da na primer pešak se množi sa 1, vitez i konj se množe sa 3, top sa 5 a kraljica sa 9. A može se uzeti u obzir i raspored figura, pa ako je top ugrožen onda delimo vrednost sa nekim brojem, itd.

Matematički, ovakva funkcija evaluacije se može predstaviti kao linearna funkcija:

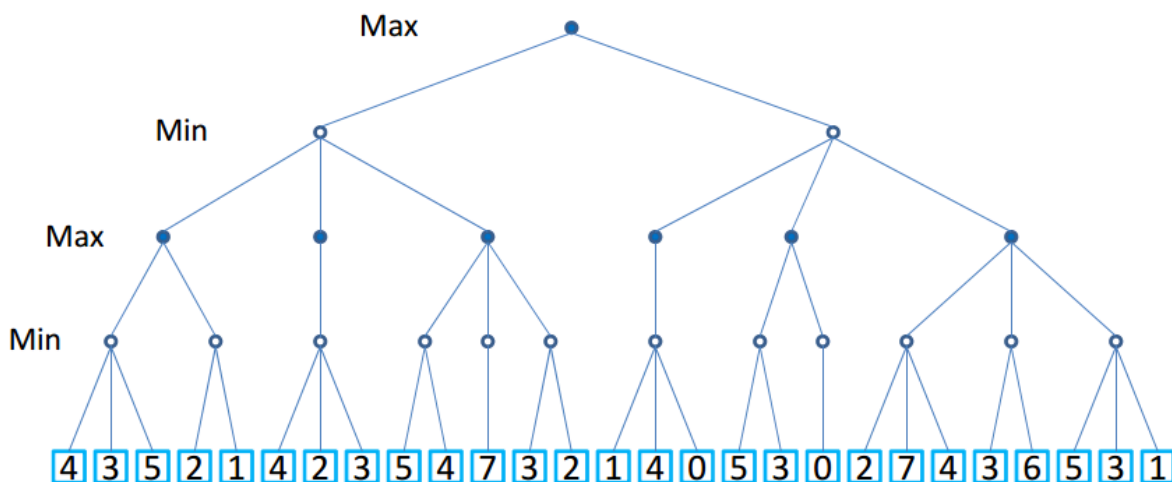
$$EVAL(s) = \omega_1 f_1(s) + \omega_2 f_2(s) + \dots + \omega_n f_n(x)$$

Gde f_i može biti broj koliko određenih figura ima na tabli, a ω_i vrednost koeficijenta za figuru.

Metod pretraživanja do određene dubine ima određene nedostatke, a glavni je taj da neka obećavajuća putanja na određenoj dubini kasnije vodi u lošu situaciju. Na primer, u šahu, ako imamo situaciju da crni igrač pretražuje do 6-og nivoa i dolazi do zaključka da će iz trenutne pozicije izgubiti kraljicu na šestoj dubini. Takođe, pretpostavimo da postoji potez u kom on žrtvuje topa na nekoj dubini koja je manja od 6 a da se na tom podstablu kraljica žrtvoje na osmoj dubini. Ovo je gora opcija jer ćemo izgubiti i topa i kraljicu, ali će algoritam izabrati taj potez jer je posmatrao samo dubinu 6. Ovo se naziva efekat horizonta (horizon effect). Kažemo da je gubitak kraljice stavljen iza horizonta pretraživanja. Ovo se može delimično prevazići menjanjem dubine pretraživanja u toku igre (ako nešto izgleda suviše dobro, pogledaj malo dublje).

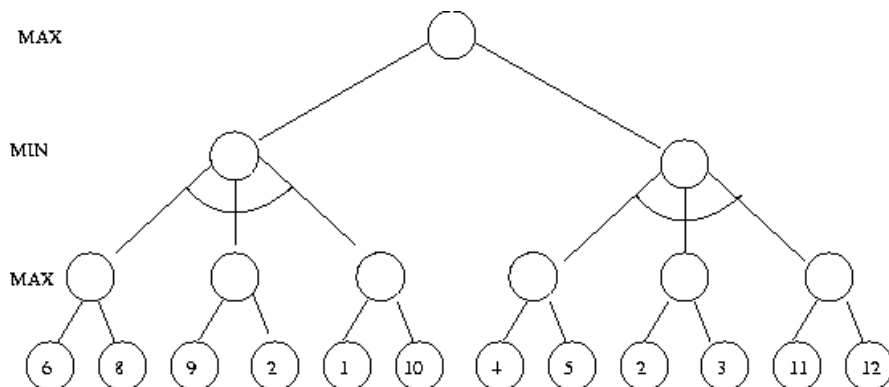
Zadaci

5.1 Odrediti vrednosti MINIMAX algoritma u svim čvorovima na stablu prikazanom na slici 5.7. Koje grane će biti odsečene Alfa-beta odsecanjem?



Slika 5.7

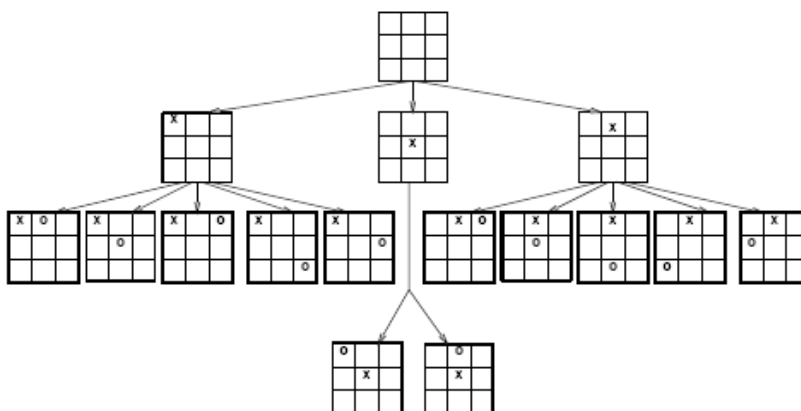
5.2. Koji grane će biti odsečene alfa-beta odsecanjem u MINIMAX stablu na slici 5.8?



Slika 5.8

5.3. Posmatrajmo igru iks-oks na tabli dimenzije 3X3. Uvodimo oznaku X_n za broj vrsta, kolona i dijagonala sa tačno n ikseva i bez nula. Slično, uvodimo oznaku O_n za broj vrsta i kolona u kojima ima n nula i nema ikseva. Završna stanja ćemo označiti sa 1 ako je $X_3 > 0$, -1 ako je $O_3 > 0$, inače završno stanje označavamo sa 0. Stanja koja nisu završna procenjujemo heurističkom funkcijom $Heuristic(s) = 3 \cdot X_2(s) + X_1(s) - (3 \cdot O_2(s) + O_1(s))$. Na slici 4.8 prikazano je stablo pretraživanja u ovoj igri do dubine 2 u kom su izostavljena simetrična stanja.

- Koja je vrednost heurističke funkcije za svaki list.
- Koristeći MINIMAX algoritam popuniti vrednosti heurističkih funkcija za čvorove na dubini 1 i 0 i označite koji potez je najbolji prema MINIMAX algoritmu.
- Koje grane će se odseći ako koristimo algoritam sa alfa-beta odsecanjem.



Slika 5.8

Poglavlje 6

Agenti zasnovani na znanju

Centralna komponenta agenta zasnovanog na znanju je **baza znanja** (knowledge base - KB). Baza znanja je skup iskaza. Svaki iskaz je napisan u jeziku za reprezentaciju znanja i predstavlja neki podatak o svetu. Neke iskaze definišemo kao aksiome, a to su iskazi koji su sami po sebi tačni i ne izvodimo ih iz drugih iskaza.

Mora postojati način da se u bazu znanja dodaju novi iskazi i da se nad bazom znanja postavljaju upiti (ove dve operacije se često označavaju sa *Tell* i *Ask*). Obe ove operacije mogu da uključe izvođenje, odnosno kreiranje novih iskaza na osnovu postojećih.

Kada se pozove agentski program, on radi tri stvari, prvo "kaže" bazi znanja koju informaciju je dobio preko senzora, zatim "pita" bazu znanja koju akciju treba da uradi. Proces pribavljanja odgovora na ovo pitanje se sastoji od procesa rezonovanja na osnovu trenutnog stanja sveta i analize mogućih ishoda akcija. Na kraju, agentski program "govori" agentu koja akcija je izabrana i agent izvršava akciju.

Agent zasnovan na znanju nije proizvoljan program za izračunavanje akcija. On koristi opis na nivou znanja, na kom specificiramo šta agent zna i koji su mu ciljevi, bez ulaženja u detalje o implementaciji.

Agent zasnovan na znanju se može napraviti tako što mu jednostavno "kažemo" sve što treba da zna. Počevši od prazne baze znanja, dizajner agenta može da mu dodaje iskaze, sve dok agent ne dobije dovoljno znanja da može da funkcioniše u nekom okruženju. Ovo se naziva deklarativni pristup razvoja sistema, za razliku od proceduralnog u kome se ponašanje implementira direktno u programskom kodu.

6.1 Svet Vumpusa

Kreiranje agenta zasnovanom na znanju biće ilustrovano na primeru sveta vumpusa (slika 6.1). Taj svet je pećina koja se sastoji iz povezanih prostorija. U jednoj prostoriji je čudovište koje se naziva vumpus i ono čeka da pojede svakog ko uđe u njegovu prostoriju. U nekim prostorijama nalazi se provalija u koju će upasti bilo ko ko uđe u tu prostoriju (osim vumpusa koji je preveliki da bi upao). U jednoj prostoriji nalazi se zlato. Iako jednostavan, ovaj svet ilustruje neke značajne koncepte inteligencije.

Ovaj svet se može precizno opisati na sledeći način (PEAS):

Mera učinka: +1000 za izlazak iz pećine sa zlatom, -1000 za ulazak u sobu u kojoj je vumpus ili one u kojima je provalija. -1 za svaku akciju, a -10 za upotrebu strele. Igra se završava kada agent pogine ili izađe iz pećine.

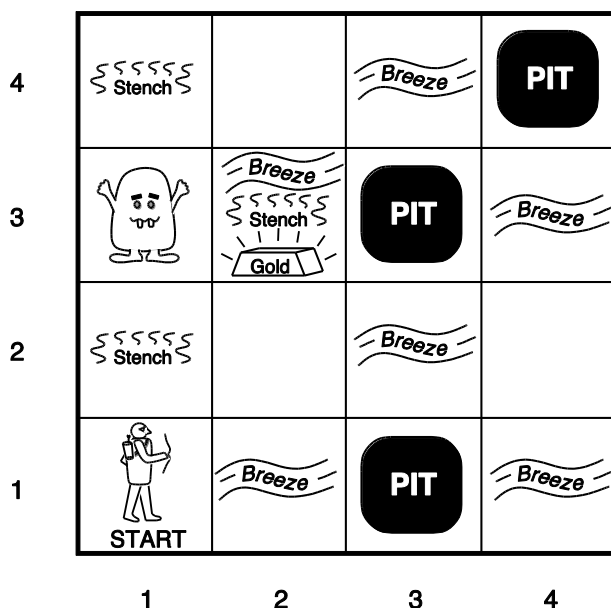
Okruženje: Mreža prostorija dimezije 4X4. Agent uvek počinje u sobi [1,1] i okrenut je nadesno. Lokacija zlata i vumpusa su proizvoljne, samo ne mogu biti početna pozicija agenta. Svako polje osim početnog može imati provaliju sa verovatnoćom 0.2.

Aktuatori: Agent može da ide napred, da se okrene u levo ili desno za 90° stepeni (*Forward*, *TurnLeft*, *TurnRight*). Agent umire ako uđe u prostoriju u kojoj je provalija ili živ vumpus. Ako agent pokuša da krene napred i naiđe na zid onda se on ne pomera. Akcija *Grab* predstavlja uzimanje zlata i može se pozvati ako je agent u istoj prostoriji gde i zlato. Akcija *Shoot* predstavlja ispaljivanje strele ravnom linijom u pravcu u kom je okrenut agent. Strela putuje dok ili ne udari i ubije vumpusa ili udari u zid. Agent ima samo jednu strelu, što znači da se akcija *Shoot* može pozvati samo jednom. Na kraju akcija *Climb* se može pozvati u prostoriji [1,1] i predstavlja izlazak iz pećine.

Senzori: Agent ima 5 senzora preko kojih može da dobije 5 informacija:

- u prostoriji u kojoj je vumpus i u susednim prostorijama (ne po dijagonali) on može da oseti neprijatan miris (*Stench*)
- u prostirajama susednim provaliji agent može da oseti povetarac (*Breeze*)
- u prostoriji u kojoj je zlato agent će videti isijavanje (*Glitter*)
- kada naiđe na zid osetiće udarac (*Bump*)
- kada usmrti vumpusa čuće vrisak (*Scream*) bilo gde u pećini.

Zapažanje agenta se može predstaviti uređenom petorkom. Na primer [Stench, Breeze, None, None, None] znači da je agent osetio neprijatan miris i povetarac, a nije video isijavanje zlata, nije udario u zid i nije čuo vrisak.



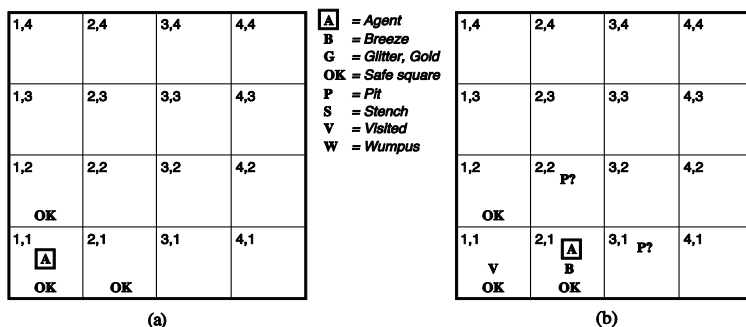
Slika 6.1

Ovo okruženje je diskretno, statično, ima jednog agenta. Ono je takođe sekvencijalno i ne može se potpuno sagledati.

Za agenta koji se nalazi u ovakvom okruženju, glavni izazov je njegovo nepoznavanje konfiguracije okruženja, koje zahteva neko logičko rezonovanje. Većina instanci vumpus sveta je takva da agent može bezbedno da pokupi zlato, ali postoje i one u kojima je to nemoguće, kao i one koje zahtevaju određeni rizik.

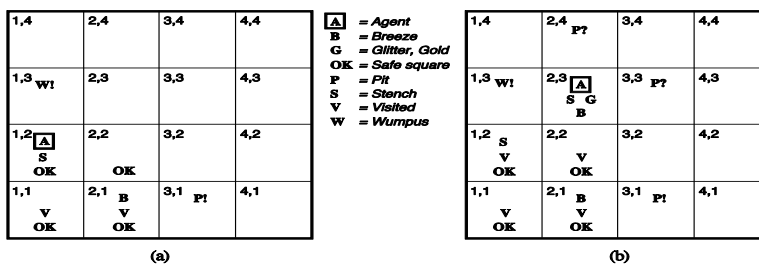
Posmatrajmo agenta u jednoj instanci sveta vumpusa koji je prikazan na slici 6.1. Agentovo inicijalno znanje o svetu sadrži pravila okruženja i on zna da je polje [1,1] u kom se on nalazi bezbedno.

Prvo zapažanje je [None, None, None, None, None] iz koga agent može da zaključi da su bezbedna polja [1,2] i [2,1] (slika 6.2a). Recimo da agent odluči da ode pravo u prostoriju [2,1]. U toj prostoriji agent oseća povetarac što znači da se u susednoj prostoriji nalazi provalija. Provalija ne može biti u prostoriji [1,1] prema pravilima igre, što znači da mora biti u prostoriji [2,2] i/ili [3,1] (slika 6.2b). U ovom momentu agent zna da je samo jedno susedno polje bezbedno, a to je [1,1]. Ako ne želi da rizikuje, jedina razumna akcija koju agent ima na raspolaganju je da se okrene i ode u polje [1,1] i zatim u polje [1,2].



Slika 6.2

U polju [1,2] agent zapaža neprijatan miris (Slika 6.3a). Neprijatan miris znači da se vumpus nalazi u jednom od susednih prostorija. Vumpus ne može biti u polju [1,1] po pravilima igre i ne može biti u polju [2,2] jer bi agent osetio neprijatan miris kada je bio u polju [2,1]. dakle, agent može da izvede zaključak da je vumpus u polju [1,3]. Takođe, pošto u polju [1,2] nema povetarca agent može da zaključi da susedno polje [2,2] nema provaliju. Kako je ranije bio rezon da je provalija ili u polju [2,2] ili u polju [3,1] agent može da zaključi da je provalija u polju [3,1]. Agent sada zaključuje da mu je bezbedno polje [2,2] i odlučuje se za tu akciju.



Slika 6.3

Rezon koji je ovde opisan je izveden na osnovu "ljudske" inteligencije. Da bi se ovakav rezon mogao automatizovati i implementirati kao veštačka inteligencija potrebno je usvojiti mehanizam za formalnu reprezentaciju znanja i izvođenje novog znanja iz postojećeg. Odgovor na ovo je matematička logika.

6.2 Logika

Logika predstavlja formalni jezik za opis informacija tako da se iz tog opisa mogu izvući neki zaključci i ona predstavlja osnovu za izgradnju baze znanja.

Rekli smo da bazu znanja čine iskazi koji predstavljaju neke informacije o svetu oko nas. Ovi iskazi su napisani u nekoj sintaksi koju definiše jezik za reprezentaciju znanja. Pored sintakse, potrebna nam je i semantika koja iskazima dodeljuje istinitosnu vrednost (tačno ili netačno; true ili false). Interpretacija je preslikavanje skupa iskaza na skup {true, false} odnosno dodela vrednosti tačno/netačno svim iskazima.

Da bi smo vršili logičko rezonovanje potreban nam je pojam logičke posledice. Kažemo da je iskaz α logička posledica baze znanja KB ako i samo ako u svakoj interpretaciji u kojoj su svi iskazi u KB tačni sledi da je i α tačno. Ovo obeležavamo sa $KB \models \alpha$.

Da bismo na osnovu neke baze znanja došli do logičke posledice potrebno nam je neko logičko rezonovanje. Logičko rezonovanje uključuje pravila izvođenja. Ako pravilo izvođenja obeležimo sa i, možemo napisati $KB \vdash \alpha$, što znači da se iskaz α može izvesti iz KB korišćenjem pravila izvođenja i.

Za pravilo izvođenja i kažemo da je saglasno ili neprotivurečno (sound) ako važi da ako je $KB \vdash \alpha$ važi da je $KB \models \alpha$, što znači da se tim pravilom izvođenja izvode samo logičke posledice.

Za pravilo izvođenja i kažemo da je kompletno ako važi da ako je $KB \models \alpha$, tada je $KB \vdash \alpha$, odnosno da se pravilom izvođenja i mogu izvesti sve logičke posledice.

6.3 Iskazna logika

Sintaksu iskazne logike čine iskazni simboli (P, Q, A, B, $W_{2,3}$) i logički veznici $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. Iskazne formule su iskazi i iskazi kombinovani sa logičkim veznicima.

Mogla bi se napraviti jednostavna baza znanja u iskaznoj logici koja opisuje svet vumpusa. Uvodimo sledeće iskazne simbole:

- ▶ $P_{x,y}$ - tačno ako je provalija u polju [x,y]
- ▶ $V_{x,y}$ - tačno ako je vumpus u polju [x,y]
- ▶ $B_{x,y}$ - tačno ako je povetarac u polju [x,y]
- ▶ $S_{x,y}$ - tačno ako je neprijatan miris u polju [x,y]

Na osnovu pravila igre imamo sledeće formule:

$$R_1 : \neg P_{1,1}$$

$$R_2 : B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$

$$R_3 : B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$

Agent je preko svojih senzora dobio neka zapažanja koje možemo predstaviti sledećim iskazima:

$$R_4 : \neg B_{1,1}$$

$$R_5 : B_{2,1}$$

Izvođenje u iskaznoj logici

Da bi smo razumeli postupak izvođenja pre toga treba uvesti neke termine. Prvi termin su logičke ekvivalencije: dve logičke formule su ekvivalentne, ako su tačne u istim interpretacijama. Na slici 6.4 prikazan je skup najčešće korišćenih logičkih ekvivalencija.

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	commutativity of \wedge
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	commutativity of \vee
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	associativity of \wedge
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	associativity of \vee
$\neg(\neg\alpha) \equiv \alpha$	double-negation elimination
$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$	contraposition
$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$	implication elimination
$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$	biconditional elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$	De Morgan
$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$	De Morgan
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	distributivity of \wedge over \vee
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	distributivity of \vee over \wedge

Slika 6.4

Drugi koncept je valjanost. Iskazna formula je valjana ako je tačna u svakoj interpretaciji. Valjane formule zovemo još i tautologije. Primer jedne valjane formule je $P \vee \neg P$.

Iz definicije logičke posledice može se izvesti teorema dedukcije:

Za svaku formulu α i β važi da je $\alpha \models \beta$ ako i samo ako važi da je $\alpha \Rightarrow \beta$ valjana formula.

To znači da možemo da pokažemo da li je $\alpha \models \beta$ tako što pokažemo da je $\alpha \Rightarrow \beta$ tautologija.

Poslednji koncept je zadovoljiva formula. Formula je zadovoljiva ako postoji interpretacija u kojoj je ona tačna. Valjanost i zadovoljivost su u vezi. α je valjana formula ako $\neg\alpha$ nije zadovoljiva. Takođe, α je zadovoljiva ako $\neg\alpha$ nije valjana.

Možemo reći i sledeće:

$\alpha \not\models \beta$ ako i samo ako formula $\alpha \wedge \neg \beta$ nije zadovoljiva. Ova tvrdnja je osnova za dokazivanje da li je neka formula logička posledica baze znanja i naziva se dokaz kontradikcijom.

Pravila izvođenja u iskaznoj logici

Pravila izvođenja daju izračunljiv način da se proverí da li je neka formula logička posledica nekog skupa formula. Ono predstavlja sredstvo za konstrisanje novih formula iz skupa postojećih.

Najpoznatija pravila izvođenja u iskaznoj logici su Modus Ponens i eliminacija veznika i.

Modus ponens:

$$\frac{\alpha \Rightarrow \beta, \alpha}{\beta}$$

Eliminacija operacije i

$$\frac{\alpha \wedge \beta}{\alpha}$$

Sve logičke ekvivalencije prikazane na slici 5.4 mogu se koristiti kao pravila izvođenja.

Dokazi se mogu izvoditi ručno ali se može primeniti i neki od algoritama pretraživanja koji bi dao sekvencu koraka (pravila) koja čine dokaz. Treba samo definisati problem dokazivanja kao problem pretraživanja:

Početno stanje: inicijalna baza znanja

Akcije: skup akcija su sva pravila izvođenja koja se primenjuju na sve odgovorajuće formule

Tranzicioni model: rezultat akcija je dodavanje nove formule (koji je rezultat pravila izvođenja) u bazu znanja.

Cilj: stanje koje sadrži formulu koji je trebalo dokazati.

Još jedna važna osobina logičkih sistema je monotonost. Monotonost logičkog sistema znači da se baza znanja može samo povećavati, kako se dodaju nove činjenice. Odnosno ako je α logička posledica neke monotone baze znanja KB, tada je ona logička posledica baze znanja $KB \wedge \beta$.

Pravilo rezolucije

Sva pravila izvođenja koja smo do sada videli su saglasna, ali šta je sa kompletnošću. Jedno pravilo izvođenja koje je kompletno je rezolucija.

Prvo ćemo uvesti pojam literala i klauzule. Literal je jedan iskaz ili njegova negacija. Klauzula je disjunkcija literala. Klauzulu koja ne sadrži ni jedan literal, nazivamo praznom klauzulom.

Ako su date klauzule:

$$C_1: P_{1,1} \vee P_{2,2} \vee P_{3,1}$$

$$C_2: \neg P_{2,2}$$

Iz njih možemo zaključiti da važi

$$C_3: P_{1,1} \vee P_{3,1}$$

Ovo se naziva pravilo rezolucije.

Klauzula C_3 je rezolvent za klauzule C_1 i C_2 . Literali koji su se poništili (jedan je negacija drugog) nazivaju se komplementarni literali.

Pravilo rezolucije predstavlja osnovu za formiranje familije kompletnih pravila izvođenja.

Konjektivna normalna forma

Pravilo rezolucije primenjuje se na klauzule, odnosno na disjunkciju literala, što znači da efikasno upotreba ove metode zahteva da baza znanja i upiti budu sastavljeni od klauzula. Postoji tvrđenje koje kaže da se svaka formula iskazne logike može napisati kao konjunkcija klauzula. Za formulu koja je napisana kao konjunkcija klauzula kažemo da je u konjuktivnoj normalnoj formi (KNF).

Postoji procedura kojom se formule mogu prevesti u KNF. Posmatrajmo primer formule $B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$. Koraci su sledeći:

1. Eliminišemo ekvivalenciju zamenjujući je sa dve implikacije

$$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge (P_{1,2} \vee P_{2,1} \Rightarrow B_{1,1})$$

2. Eliminišemo implikaciju zamenjujući $\alpha \Rightarrow \beta$ sa $\neg \alpha \vee \beta$.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg (P_{1,2} \vee P_{2,1}) \vee B_{1,1})$$

3. KNF zahteva da negacije bude uz literal, što postizemo korišćenjem demorganovih pravila i eliminacijom duple negacije.

4. Na kraju još samo treba primeniti pravilo distributivnosti operacije \vee nad operacijom \wedge i dobija se formula koja je u konjuktivnoj normalnoj formi.

$$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (P_{2,1} \vee B_{1,1}).$$

Metod rezolucije

Pravilo izvođenja zasnovano na rezoluciji koristi pokaz kontradikcijom. Njime se pokazuje da važi $KB \vdash \alpha$ tako što se pokaže da $KB \wedge \neg \alpha$ nije zadovoljiva, odnosno da vodi u kontradikciju. Za ovo se koristi metod rezolucije.

Prvi korak je da se $KB \wedge \neg \alpha$ prevede u konjuktivnu normalnu formu. Zatim se pravilo rezolucije primenjuje na rezultujuće klauzule, čime se dobijaju nove klauzule. Proces se nastavlja dok se ne desi jedan od dva slučaja. Prvi je da više nemamo komplementarnih literala u kom slučaju zaključujemo da α nije logička posledica KB. Drugi slučaj je da dobijemo prazna klauzulu i tada zaključujemo da α jeste logička posledica od KB.

6.5 Logika prvog reda

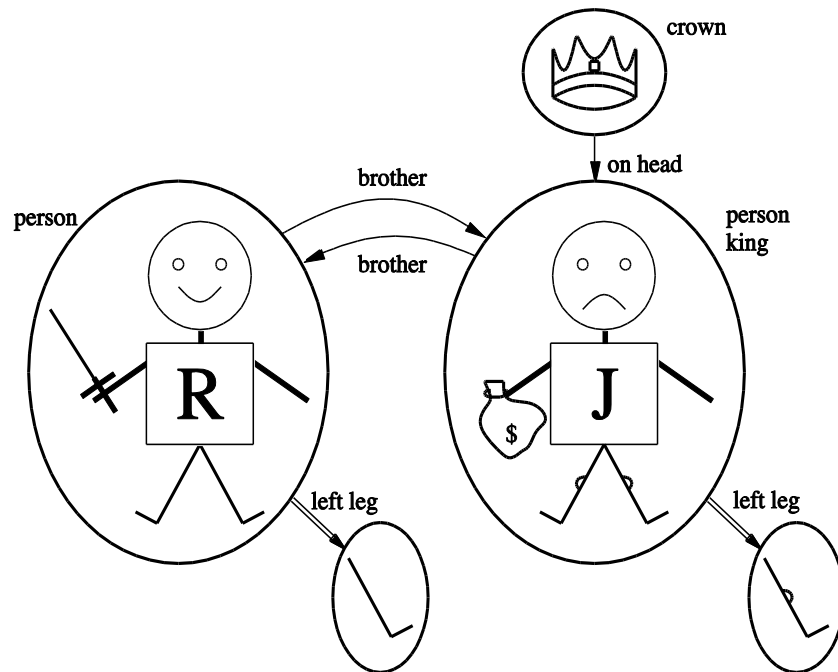
Programski jezici (kao što su Java, C++ ili Lisp) su danas najčešće korišćeni formalni jezici. Programi koji su u njima napisani predstavljaju neke procese izračunavanja i podržavaju strukture podataka kojima se mogu predstaviti činjenice o svetu (za predstavljanje činjenica razvijene su i baze podataka). Međutim, ono što nedostaje tim programskim jezicima je neki uopšteni mehanizam za izvođenje novih činjenica iz postojećih. Svako ažuriranje strukture podataka u njima uslovljeno je procedurom koja je specifična za taj problem i koju implementira programer na osnovu svog znanja o problemu. Ovaj proceduralni pristip može se zameniti deklarativnim pristupom koji ima iskazna logika. Međutim, iskazna logika ima slabu izražajnu snagu da koncizno opiše okruženje koje ima više objekata.

Na primer, za uslov u vumpus svetu da postoji povetarac u prostoriji koja je susedna onoj u kojoj se nalazi provalija, morali smo da pravimo formule za svaku prostoriju ($B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$). Bilo bi mnogo bolje kad bismo mogli da kažemo „U prostorijama koje su pored provalije oseća se povetarac“. Ova rečenica je napisana prirodnim jezikom.

Logika prvog reda koja se naziva još i predikatski račun prvog reda je dovoljno izražajni jezik da opiše dobar deo našeg znanja o svetu. Ona kombinuje jezik iskazne logike sa prirodnim ljudskim jezikom. Ako analiziramo prirodni jezik videćemo da on ima neke imenice i imeničke izraze koji se odnose na neke objekte (prostorija, provalija, vumpus). Takođe prirodni jezik ima glagole i glagolske izraze koji opisuju neke relacije (je susedan, usmrtniti, kretati se...). Neke od ovih relacija su funkcije – relacije koje imaju samo jednu vrednost za neki ulaz (otac, najbolji prijatelj,...). Relacijama se mogu definisati i neke osobine (crveno, okruglo,...) ili neki odnosi (brat, veće, unutar, deo, između,...).

Sintaksa i semantika logike prvog reda

Kod iskazne logike smo imali preslikavanje iskaza na skup {true, false}. Ovde su stvari mnogo interesantnije, imamo preslikavanje da neki domen, koji predstavlja skup objekata. Potrebno je da domen ne bude prazan. Posmatrajmo primer domena na slici 6.5, koja prikazuje dva kralja, Ričarda (Richard) i Džona (John). Ovaj domen ima pet objekata, kralja Ričarda, kralja Džona, njihove leve noge i krunu. Objekti ovde mogu da imaju razne relacije. Na primer, Ričard i Džon su braća. Relacija nije ništa drugo nego torka nekih objekata (uređeni skup objekata). Relacija brat ima dve torke {<Ričard, Džon>, <Džon, Ričard>}. Kruna je na glavi kralja Džona. Relacija „na glavi“ ima samo jednu torku {<kruna, Džon>}. „Brat“ i „na glavi“ su binarne relacije jer povezuju par objekata. Na slici je prikazana i unarna relacija osoba (person) koja je tačna za Ričarda i Džona. Neke vrste relacija posmatramo kao funkcije, one koje preslikavaju neki objekat (ili skup objekata) na tačno jedan objekat. Primer funkcije sa slike je leva noga (left leg).



Slika 6.5

Simboli i interpretacije

Sintaksu logike prvog reda čine simboli koji označavaju objekte, relacije i funkcije. Postoje tri vrste simbola: konstante (koje predstavljaju objekte), predikatski simboli (relacije) i funkcijski simboli (funkcije). Ovde ćemo usvojiti konvenciju da sve ove simbole označavamo prvim velikim slovom (NaGlavi, Osoba, Ričard, Džon). Svaki predikat i funkcija pored naziva imaju i arnost (broj argumenata).

Kao i kod iskazne logike, i ovde moramo da odredimo iskaze koji su tačni i one koji su netačni. Ovo nam obezbeđuje interpretacija koja za simbole vezuje konkretne objekte, relacije i funkcije (na primer, relaciju Brat interpretiramo kao rodbinski odnos između dva muškarca koji imaju iste roditelje, konstantu Džon interpretiramo kao kralja Džona koji je vladao tada i tada,...).

Term je logički izraz koji opisuje neki objekat. U terme spadaju konstante, promenljive i funkcije. Konstante su termini kojima imenujemo neke objekte, a funkcije koristimo da bismo opisali neke objekte (LeftLeg(John)). Funkcije kao svoje argumente sadrže druge terme.

Atomske formule (atomi) su predikati koji sadrže listu terma kao argumente u zagradi (na primer Brother(Richard, John), Married(Father(Richard), Mather (John))). Kompleksne formule nastaju kombinacijom atomskih formula i logičkih operacija (iskazne logike).

Kvantifikatori

Kvantifikatore koristimo kad hoćemo da opišemo osobine cele kolekcije objekata. Postoje dva standardna kvantifikatora, univerzalni (\forall) i egzistencijalni (\exists). Kvantifikatori se vezuju za promenjive, koje po konvenciji pišemo malim slovima. I promenjive su vrsta terma.

Univerzalni kvantifikator, \forall se obično čita kao „za svako“. Na primer, formula $\forall x \text{ Kralj}(x) \Rightarrow \text{Osoba}(x)$ se čita kao „za svako x važi da ako je x kralj onda je x osoba“. Izraz $\forall x P$ označava da je P tačno za svaki element domena na koji se odnosi x . Univerzalni kvantifikator se najčešće koristi sa implikacijom.

Egzistencijalni kvantifikator, \exists se čita kao „postoji“. On se koristi kada hoćemo da kažemo da za neke objekte nešto važi. Na primer ako hoćemo da kažemo da kralj Džon ima krunu na glavi, formula za to je $\exists x \text{ Kruna}(x) \wedge \text{NaGlavi}(x, \text{Džon})$. Ova formula govori da postoji neko x za koje je tačno da je to x kruna i da je ono na Džonovoj glavi. Egzistencijalni kvantifikator se najčešće koristi sa konjukcijom.

U logici prvog reda možemo koristiti znak jednakosti kad hoćemo da kažemo da se dva terma odnose na isti objekat. na primer, $\text{Otac}(\text{Džon}) = \text{Henri}$. Može se koristiti i sa negacijom ($\neg(x=y)$) kada hoćemo da kažemo da se dva terma ne odnose na isti objekat. Često se koristi i skraćeni zapis $x \neq y$.

U logiku prvog reda, uvedene su neke konvencije koje omogućavaju jednostavnije predstavljanje znanja. Prva je pretpostavka o jedinstvenosti imena koja kaže da se konstante koje su različite (imaju različito ime) odnose na različite objekte. Druga konvencija je da svaku atomsku formulu za koju nije navedeno da je tačna tumačimo kao netačnu. I poslednja je zatvorenost domena koja znači da domen sadrži samo one elemente koji su navedeni kao konstante.

Korišćenje logike prvog reda

Rečenice logike prvog reda (termi i formule) se dodaju u bazu znanja korišćenjem operacije Tell. Ovakve rečenice se nazivaju tvrdnje. Na primer:

Tell(KB, Kralj(Džon)).

Tell(KB, Osoba(Ričard)).

Tell(KB, $\forall x \text{ Kralj}(x) \Rightarrow \text{Osoba}(x)$).

Bazi znanja možemo da postavimo pitanje operacijom Ask:

Ask(KB, Kralj(Džon)).

Odgovor na ovo pitanje je true (odnosno tačno). Pitanja koja se na ovaj način postavljaju nazivaju se upiti ili ciljevi.

Možemo da postavimo upite sa kvantifikatorima

Ask(KB, $\exists x$ Osoba(x)).

Ako hoćemo da saznamo za koje x je nešto tačno pitaćemo sledeće:

AskVars(KB, Person(x)).

Kao rezultat ovakvog upita dobijamo niz odgovora. U ovom slučaju biće dva odgovora {x/Džon} i {x/Ričard}.

Primer domena: kraljevska porodica

Objekti u domenu su ljudi.

Unarni predikati su *Musko*, *Zensko*.

Porodični odnosi su binarni predikati: *Roditelj*, *Otac*, *Majka*, *Deda*, *Sestra*, *Dete*, *Sin*, *Kći*, *Unuk*,...

Koristićemo funkcije *Majka* i *Otac* jer pretpostavljamo da svako ima samo jednu majku i jednog oca.

Neke od formula baze znanja su:

$$\forall m, d \text{ Majka}(d) = m \Leftrightarrow \text{Zensko}(m) \wedge \text{Roditelj}(m, d)$$

$$\forall m, d \text{ Otac}(d) = o \Leftrightarrow \text{Musko}(o) \wedge \text{Roditelj}(o, d)$$

$$\forall m, z \text{ Muz}(m, z) \Leftrightarrow \text{Musko}(m) \wedge \text{Supruznik}(m, z)$$

$$\forall x \text{ Musko}(x) \Leftrightarrow \neg \text{Zensko}(x)$$

$$\forall x, y \text{ Roditelj}(x, y) \Leftrightarrow \text{Dete}(y, x)$$

$$\forall x, y \text{ Deda}(x, y) \Leftrightarrow \exists p \text{ Otac}(x, p) \wedge \text{Roditelj}(p, y)$$

...

Sve ove formule mogu se posmatrati kao aksiome domena kraljevske porodice. Aksiomama se predstavljaju neke osnovne informacije iz kojih se mogu izvući neki zaključci - teoreme. Aksiome koje imaju oblik $\forall x, y P(x, y) \Leftrightarrow \dots$ nazivaju se definicije. Aksiome mogu biti i jednostavne činjenice tipa *Musko(Petar)*, *Supruznik(Petar, Milena)*.

6.6 Inženjerstvo znanja

Proces konstruisanja baze znanja naziva se inženjerstvo znanja. Postoji tzv. baza znanja za specifični svrhe (special-purpose) čiji domen je ograničen i svi upiti su unapred poznati. Ovde ćemo se baviti ovakvim bazama znanja. Pored toga postoje i generalne baze znanja (general purpose) koje pokrivaju široki skup ljudskog znanja i namenjene su zadacima kao što su razumevanje prirodnog jezika i sl.

Projekti inženjerstva znanja mogu da se razlikuju u sadržaju, opsegu i složenosti, ali se svaki takav projekat sastoji od sledećih koraka

1. Identifikacija zadatka

U ovom delu identifikuju se sva moguća pitanja na koje baza znanja treba da da odgovor, kao i vrste činjenica koje će biti dostupne u različitim instancama problema. Na primer, da li će vumpus baza znanja davati odgovore na pitanja o tome koju akciju treba izvršiti ili će samo da daje informacije o okruženju. Identifikacija zadatka određuje koje znanje treba da se reprezentuje da bi se instance problema povezale se odgovorima.

2. Prikupljanje relevantnog znanja

U ovom delu radi se sa ekspertima iz oblasti za koju se projektuje baza znanja da bi se prikupili svi relevantni podaci za kreiranje baze znanja. Ovaj proces se naziva usvajanje znanja (knowledge aquisition). U ovom delu znanje se ne predstavlja formalno već se samo razmatra opseg baze znanja i način na koji ono funkcioniše.

3. Određivanje rečnika predikata, funkcija i konstanti

Ovo je korak koji prevodi važne koncepte domena u logičke koncepte. Od ovog dela u velikoj meri zavisi uspeh projekta. Ovde se odgovara na pitanja tipa, da li provalije predstaviti kao objekte ili kao unarne predikate? Da li orijentacija agenta treba da bude funkcija ili predikat? Da li lokacija vumpusa treba da zavisi od vremena? Kada se donesu ove odluke, kao rezultat dobijamo rečnik koji se naziva ontologija domena.

4. Kodiranje uopštenog znanja o domenu korišćenjem usvojene ontologije

Pišu se aksiome za sve koncepte ontologije. Ovde se terminima usvojenim u 3 daje nekakvo formalno značenje. U ovom delu moguće je uočiti neke nedostatke rečnika koji uslovljavaju vraćanje na tačku tri da bi se popravila ontologija.

5. Kodiranje znanja o specifičnim instancama problema

Ako je ontologija dobro formirana ovaj deo je jednostavan i on uključuje pisanje jednostavnih atomskih rečenica o instancama konceptata koji su deo ontologije. Za logičke agente, deo podataka o instancama problema se dobija preko senzora. Za bazu znanja je ovo analogija se ulaznim podacima u proceduralnim programima.

6. Postavljanje upita i dobijanje odgovora

Ovde pozivamo procedure zaključivanja nad aksiomama i činjenicama baze znanja da bismo dobili nove činjenice i teoreme. Na ovaj način smo izbegli pisanje algoritama specifičnih za problem.

7. Debugiranje baze znanja

Ovaj deo je testiranje baze znanja postavljanjem upita i analizom odgovora da bi se utvrdile eventualne greške.

6.7 Zaključivanje u logici prvog reda

Zaključivanje u logici prvog reda je osnova za sve moderne logičke sisteme.

Posmatrajmo sledeći skup rečenica predikatskog računa

$$\forall x \text{ Kralj}(x) \wedge \text{Pohlepan}(x) \Rightarrow \text{Zao}(x)$$

$$\text{Kralj}(\text{Džon})$$

$$\text{Pohlepan}(\text{Džon})$$

Iz ovog skupa činjenica se očigledno može zaključiti da važi $\text{Zao}(\text{Džon})$. To je očigledno čoveku, ali se može napraviti i generealno pravilo koje ovakvo zaključivanje može napraviti očiglednim i računaru.

Proces zaključivanja da je Džon zao, odnosno da je $\{x/\text{Džon}\}$ rešenje upita $\text{Zao}(x)$ se odvija na sledeći način: da bismo koristili pravilo da su pohlepni kraljevi zli, potrebno je naći neko x takvo da je x kralj i da je x pohlepan i tada možemo zaključiti da je taj x i zao. Uopšteno govoreći, ako postoji supstitucija θ kojom se klauzule sa leve strane implikacije (povezane sa \wedge) mogu izjednačiti sa nekom činjenicom iz baze znanja tada možemo izvesti zaključak koji je sa desne strane implikacije primenom supstitucije θ . U ovom slučaju supstitucija je $\theta = \{x/\text{Džon}\}$.

Moguća su i složenija zaključivanja. Na primer, pretpostavimo da umesto $\text{Pohlepan}(\text{Džon})$ imamo

$$\forall y \text{ Pohlepan}(y).$$

Tada opet možemo zaključiti da je Džon zao, jer znamo da je Džon kralj i znamo da je pohlepan (jer je svako pohlepan). U ovom slučaju potrebno je pronaći supstituciju za promenljive u implikaciji i za promenljive u iskazima iz baze znanja. U ovom slučaju potrebna supstitucija je $\{x/\text{Džon}, y/\text{Džon}\}$ koja levu stranu implikacije $\text{Kralj}(x)$ i $\text{Pohlepan}(x)$ čini identičnim sa rečenicama iz baze znanja $\text{Kralj}(\text{Džon})$ i $\text{Pohlepan}(y)$ i možemo izvesti zaključak (desna strana implikacije).

Ovakav proces rezonovanja se naziva Generalizovani Modus Ponens.

Opisani proces zaključivanja zahteva pronalaženje zamena kojima se različiti logički izrazi mogu napraviti identičnim. Ovaj proces naziva se unifikacija i predstavlja glavni koncept rezonovanja u logici prvog reda.

Unifikacija je algoritam za određivanje zamena (supstitucija) koje treba izvršiti da bi se uparila dva izraza predikatskog računa. Zamena ili supstitucija odnosi se na promenu formule tako što se promenljive koje su uz univerzalne kvantifikatore zamenjuju sa odgovarajućim termom.

Proces unifikacije uzima dve formule predikatskog računa i vraća njihov unifikator ukoliko postoji.

$$\text{UNIFY}(p,q) = \theta \text{ gde je } \text{SUPST}(\theta,p) = \text{SUPST}(\theta,q).$$

Da bi se unifikacija primenjivala potrebno je da nemamo promenljive koje su vezane za egzistencijalne kvantifikatore. On se može zameniti u rečenicama oblika $\exists x \text{ Roditelj}(X, \text{Marko})$ sa izrazom $\text{Roditelj}(\text{Petar}, \text{Marko})$ i $\text{Roditelj}(\text{Marija}, \text{Marko})$, ako se pretpostavi da su u zadatoj interpretaciji Petar i Marija roditelji Marku. Proces eliminisanja egzistencijalnog kvantifikatora je komplikovaniji u rečenicama oblika $\forall x \exists y \text{ Majka}(x, y)$. Ovakav izraz se može zameniti sa $\forall x \text{ Majka}(x, F(x))$ gde je F skolem

funkcija od x . Proces skolemizacije zamenjuje vrednost promenjive koja stoji uz egzistencijalni kvantifikator sa konstantom koja se dobija kao vrednost funkcije promenljivih uz koje je vezan univerzalni kvantifikator.

Kako radi unifikacija nad bazom znanja? Pretpostavimo da bazi znanja postavimo upit $\text{AskVars}(\text{Poznaj}(Ana, x))$: koga poznaje Ana? Odgovor na ovo pitanje će se dobiti pronalaženjem svih rečenica u bazi znanja koje mogu da se unifikuju sa $\text{Poznaj}(Ana, x)$. Evo kako će izgledati rezultati unifikacije za tri rečenice koje se mogu naći u bazi znanja:

$\text{UNIFY}(\text{Poznaj}(Ana, x), \text{Poznaj}(Ana, Marija)) = \{x/Marija\}$

$\text{UNIFY}(\text{Poznaj}(Ana, x), \text{Poznaj}(y, Petar)) = \{x/Petar, y/Ana\}$

$\text{UNIFY}(\text{Poznaj}(Ana, x), \text{Poznaj}(y, Majka(y))) = \{y/Ana, x/Majka(Ana)\}$

Nekad možemo imati više unifikatora. Na primer u $\text{UNIFY}(\text{Poznaj}(Ana, x), \text{Poznaj}(y, z))$ imamo dve moguće unifikacije $\{y/Ana, x/z\}$ ili $\{y/Ana, x/Ana, z/Ana\}$. Duga unifikacija se može dobiti preko prve. Za ovakav slučaj kažemo da je prvi unifikator opštiji od drugog jer on stavlja manje restrikcije na promenljive. Uvek postoji jedan najopštiji unifikator.

Postoje dva jednostavna postupka zaključivanja u predikatskom računu prvog reda koja se zasnivaju na unifikaciji, to su ulančavanje unapred (forward-chaining) i ulančavanje unazad (backward-chaining). Ulančavanje unapred je specijalan slučaj opštenog koncepta koji se naziva podatak-orijentisano rezonovanje (data-driven reasoning) u kom rešavanje problema počinje od poznatih činjenica. Ulančavanje unazad je specijalan slučaj opšteg koncepta koje se naziva cilj-orijentisano rezonovanje (goal-driven reasoning) u kom se prvo uoči cilj koji treba da se dostigne, zatim pravila koja vode do cilja i određuju se uslovi koji moraju biti ispunjeni da bi se primenom tih pravila došlo do cilja. Ovi uslovi postaju novi ciljevi ili podciljevi.

Ova dva algoritma primenjuju se na bazu znanja koja se sastoji od atomaskih termova i formula koje sadrže implikaciju koja sa leve strane ima konjukciju pozitivnih literala, a sa desne pozitivan literal. Primer jedna ovakve baze znanja je

$\forall x \text{ Kralj}(x) \wedge \text{Pohlepan}(x) \Rightarrow \text{Zao}(x)$

$\text{Kralj}(Džon)$

$\text{Pohlepan}(Džon)$

Ne mogu se sve baze znanja prevesti u ovakav oblik, ali mnoge mogu i na njih se mogu primeniti ova dva algoritma zaključivanja.

Posmatrajmo sledeći problem:

Zakon kaže da je kriminalno delo za stanovnike Srbije da prodaju alkohol maloletnim licima. Dečak Nikola ide u osnovnu skolu i kod njega su pronađene flaše vina, i sve te flaše mu je prodao Petar Petrović iz Srbije.

Iz ovoga ćemo zaključiti da je Petrović kriminalac.

Prvo treba napisati bazu znanja problema:

"...da je kriminalno delo za stanovnike Srbije da prodaju alkohol maloletnim licima"

1: StanovnikSrbije(x) \wedge Alkohol(y) \wedge Prodaje(x,y,z) \wedge Maloletan(z) \Rightarrow Kriminalac(x)

" Nikola ... i kod njega su pronađene flaše vina ".

2: Posедуje(Nikola,V)

3: Vино(V).

" i sve te flaše mu je prodao Petar Petrović"

4: Vино(x) \wedge Poseduje(Nikola,x) \Rightarrow Prodaje(Petrović, x, Nikola)

Moramo da znamo da je vino vrsta alkohola

5: Vино(x) \Rightarrow Alkohol(x)

Moramo napisati uslov da su deca koja idu u osnovnu školu maloletna

6: Skola(x,Osnovna) \Rightarrow Maloletan(x)

Zatim da je Petrović iz Srbije

7: StanovnikSrbije(Petrović).

Zatim da Nikola ide u osnovnu školu

8: Skola(Nikola,Osnovna).

Ulančavnje unapred

Algoritam zaključivanja ulančavanje unapred započinje sa poznatim činjenicama i vrši supstituciju u svim implikacijama u kojima su svi literali sa leve strane mogu unifikovati sa nekom poznatom činjenicom. Desna strana implikacije se dodaje bazi znanja. I taj postupak se nastavlja sve dok se ne odgovori na upit ili dok više nema implikacija za unifikaciju.

Iskoristićemo primer prodaje alkohola da bi ilustrovali ovaj algoritam. U posmatranoj bazi znanja imamo četiri implikacije (1,4,5,6) i pošto ne mogu sve odmah da se upare imaćemo više iteracija (u ovom slučaju dve)

- u prvoj iteraciji implikacija 1 ne može da se unifikuje
implikacija (4) može da se unifikuje sa (3 i 2) supstitucijom {x/V}, u bazu znanja se dodaje Prodaje(Petrović,V,Nikola)
implikacija (5) može da se unifikuje sa (3) supstitucijom {x/V}, u bazu znanja se dodaje Alkohol(V)
implikacija (6) može da se unifikuje sa (8) supstitucijom {x/Nikola}, u bazu znanja se dodaje Maloletan(Nikola)
- U drugoj iteraciji u implikaciji (1) se sada može izvršiti supstitucija {x/Petrović}, {y/V}, {z/Nikola} i bazi znanja se dodaje Kriminalac(Petrović).

Pravilo izvođenja ulančavanje unapred je saglasno jer se zasniva na pravilu Modus Ponens (Generalizovani modus ponens) koje je saglasno. Ovo pravilo izvođenja je i kompletno za baze znanja koje sadrže atomske terme i implikacije u opisanom obliku.

Ulančavanje unazad

Ulančavanje unazad počinje od cilja i proverava da li on može da se unifikuje sa literalom koji je desna strana implikacije. Ukoliko može, u formuli se izvrši substitucija i literali koji se nalaze sa leve strane implikacije postaju novi ciljevi. Ovaj postupak se nastavlja sve dok više nema ciljeva koji nisu unifikovani.

U primeru sa prodajom alkohola postavljamo cilj Kriminalac(Petrović) i posmatramo desne strane implikacija. Implikacija (1) ima desnu stranu koja može da se unifikuje sa Kriminalac(Petrović), odgovarajuća substitucija je $\{x/\text{Petrović}\}$. Nakon ovoga dobijamo 4 podcilja, to su StanovnikSrbije(Petrović), Alkohol(y), Prodaje(Petrović,y,z) i Maloletan(z). Pošto činjenicu StanovnikSrbije(Petrović) imamo u bazi znanja ovaj podcilj je zadovoljen i sada drugi cilj tražimo na desnoj strani implikacije. Ovaj postupak se nastavlja sve dok ne zadovoljimo sve ciljeve.

6.8 Rezolucija u logici prog reda

Metod rezolucije se može proširiti na korišćenje sa logikom prvog reda.

Prvi korak rezolucije kod logike prvog reda je prevođenje predikatke formule u konjuktivnu normalnu formu.

1. Eliminacija implikacije korišćenjem logičke ekvivalencije

$$p \Rightarrow q \equiv \neg p \vee q$$

2. Uvlačenje negacije

Ako imamo negaciju sa kvantifikatorima koristimo sledeće zamene

$$\neg \forall x p \text{ postaje } \exists x \neg p$$

$$\neg \exists x p \text{ postaje } \forall x \neg p$$

Inače negaciju uvlačimo korišćenjem Demorganovih pravila.

3. Preimenovanje promenljivih

U rečenicama oblika $(\exists x P(x)) \vee (\exists x Q(x))$ vršimo preimenovanje jedne promenljive. Ovo radimo da bismo izbegli konfuziju kasnije kada skinemo kvantifikatore.

4. Skolemizacija: Eliminešemo egzistencijani kvantifikator tako što promenljive koje su za njega vezane zamenjujemo sa konstantom ili funkcijom (ukoliko imamo oblik $\forall x \exists y$).

5. Brišemo univerzalne kvantifikatore

6. Izvršimo distribuciju \vee nad \wedge .

Na ovaj način dobijamo konjuktivnu normalnu formu.

Metodom rezolucije dokazujemo da važi $KB \models \alpha$ tako što dokazujemo da $KB \wedge \neg\alpha$ nije zadovoljiva (dokaz kontradikcijom), isto kao kod iskazne logike.

Prvi korak je da se sve formule baze znanja i negacija posledice, prevedu u konjektivnu normalnu formu. Zatim se pronalaze dva komplementarna literala. U logici prvog reda dva literala su komplementarna ako jedan može da se unifikuje sa negacijom drugog. Komplementarni literali se eliminišu. A supstitucija (dobijena na osnovu unifikacije) se primenjuje na ostatak klauzule.

primer: Petrović kriminalac

6.9 Logičko programiranje

Logičko programiranje je tehnologija koja se oslanja na deklarativni pristup logike. Najpoznatiji jezik za logičko programiranje je Prolog. Prolog se najviše koristi za pisanje kompajlera i obradu prirodnog jezika. Takođe, postoje veliki broj ekspertnih sistema u oblasti finansija, prava, medicine koji su napisani u Prologu. Prolog program je skup implikacija i činjenica. Implikacije se nazivaju pravila i navode se u obrnutom redosledu, na primer logička formula $A \wedge B \Rightarrow C$ u prologu se navodi $C:-A,B$. Konjunkcija se označava zarezom, a disjunkcija tačka-zarezom.

Izvršavanje Prolog programa implementira algoritam ulančavanja unazad nad bazom znanja i na taj način se kreira stablo pretraživanja. Koren stabla pretraživanja je polazni cilj. Kreiranje stabla pretraživanja Prologa ilustrovano je u nastavku na primerima.

Primer 1

(k₁) p(X, Y):-q(X), r(X, Y)

(k₂) p(d,4)

(k₃) q(a)

(k₄) q(b)

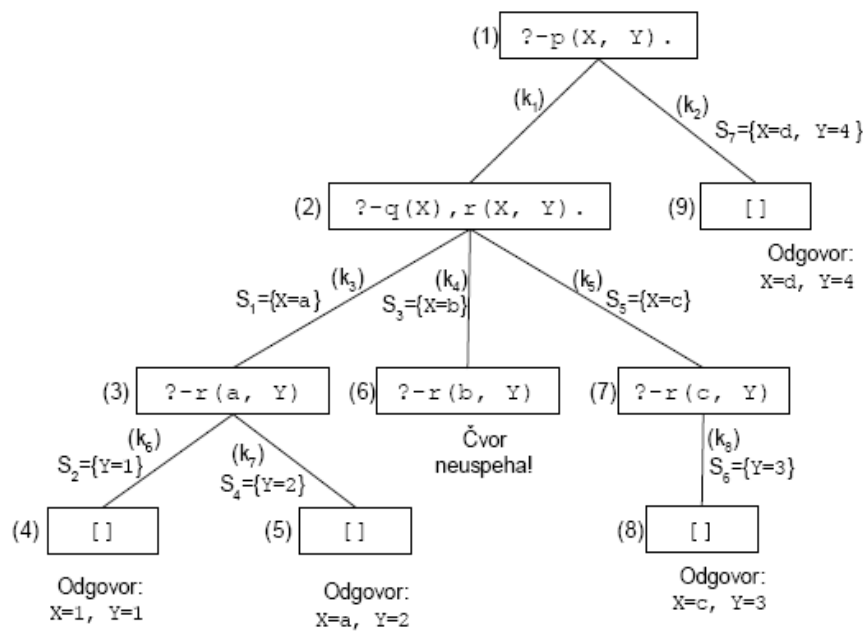
(k₅) q(c)

(k₆) r(a,1)

(k₇) r(a,2)

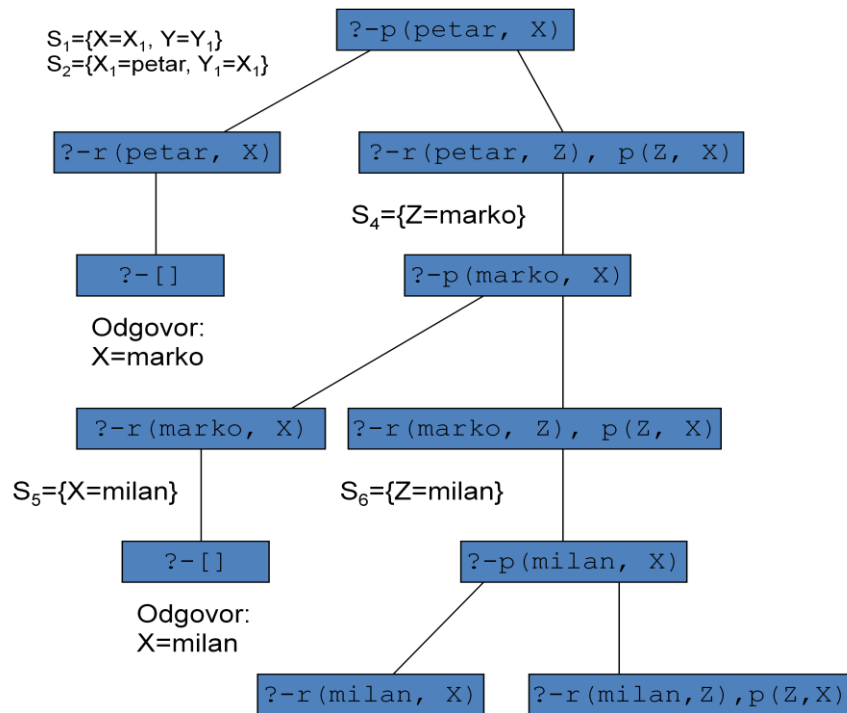
(k₈) r(c,3)

?-p(X,Y)

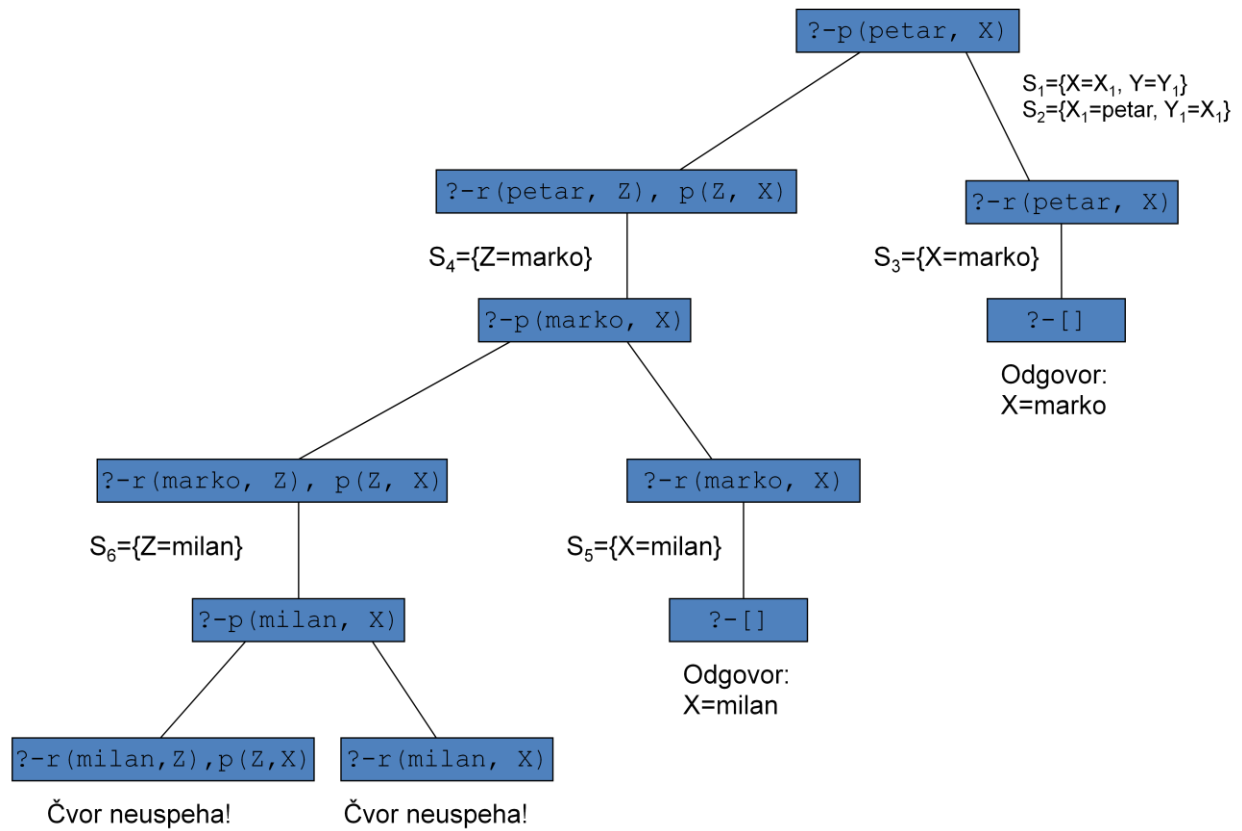


Primer 2

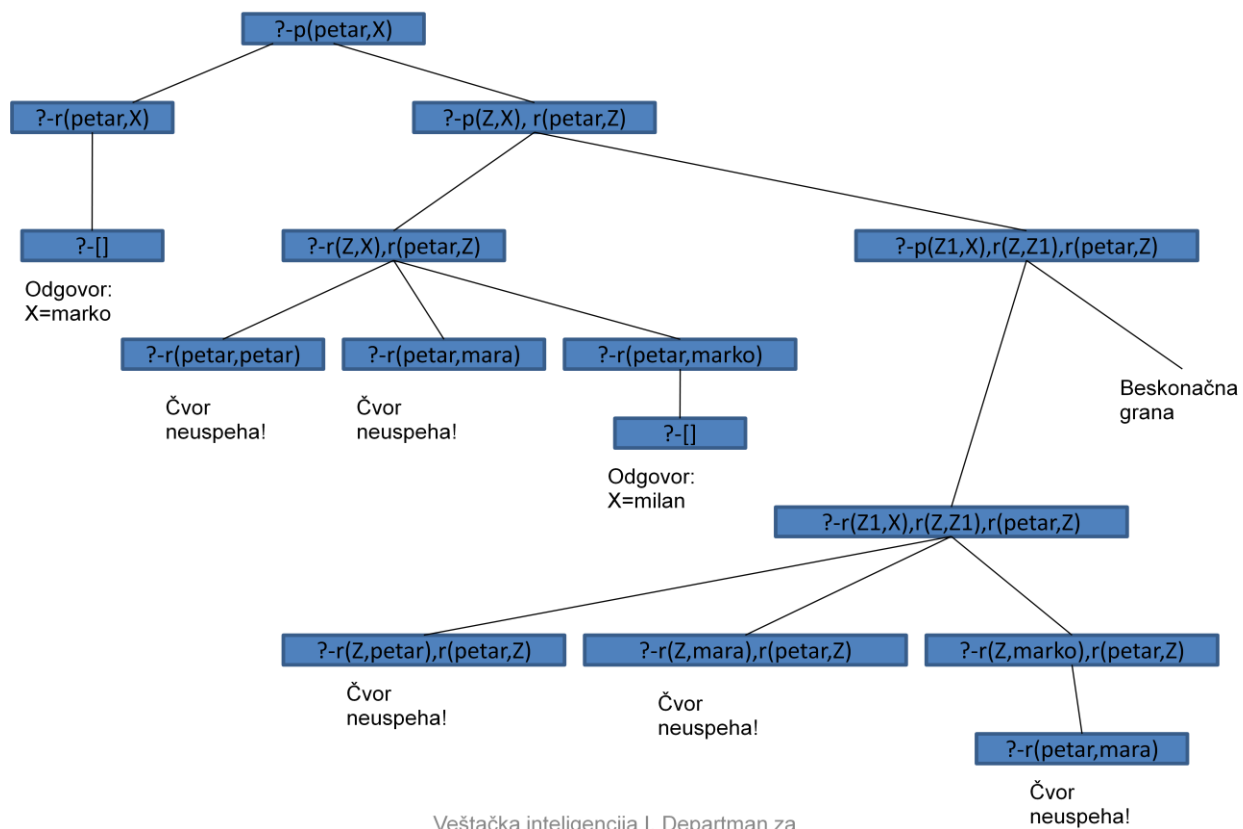
$r(\text{petar}, \text{marko}).$
 $r(\text{mara}, \text{marko}).$
 $r(\text{marko}, \text{milan}).$
 $p(X, Y) :- r(X, Y).$
 $p(X, Y) :- r(X, Z), p(Z, Y).$
 $?-p(\text{petar}, X)$



$r(\text{petar}, \text{marko}).$
 $r(\text{mara}, \text{marko}).$
 $r(\text{marko}, \text{milan}).$
 $p(X, Y) :- r(X, Z), p(Z, Y).$
 $p(X, Y) :- r(X, Y).$
 $?-p(\text{petar}, X)$



$r(\text{petar}, \text{marko}).$
 $r(\text{mara}, \text{marko}).$
 $r(\text{marko}, \text{milan}).$
 $p(X, Y) :- r(X, Y).$
 $p(X, Y) :- p(Z, Y), r(X, Z).$
 $?-p(\text{petar}, X)$



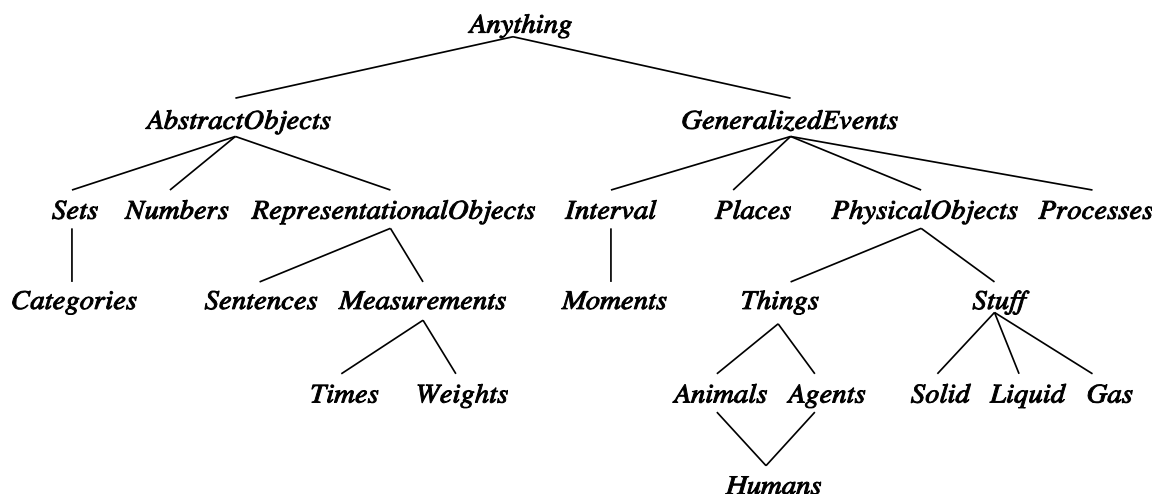
6.10. Reprezentacija znanja

Do sada smo videli tehnologije za implementaciju agenata zasnovanih na znanju: sintaksu, semantiku i rasuđivanje u logici prvog reda. U ovom odeljku bavićemo se pitanjem koji *sadržaj* staviti u bazu znanja, odnosno kako reprezentovati činjenice o svetu.

6.10.1 Ontološki inženjering

U domenima jednostavnih zadataka izbor reprezentacije znanja nije od velikog značaja. Međutim, kompleksni domeni iz realnog sveta zahtevaju uopštene i fleksibilne reprezentacije. Predstavljanjem apstraktnih koncepata kao što su Događaj, Vreme, Fizički objekti, Uverenja bavi se oblast koja se naziva ontološki inženjering (eng. ontological engineering).

Predstavljanje svega što postoji je neograničen problem, umesto toga ideja ontološkog inženjeringa je da napravi okvirnu ontologiju u koju će moći da se naknadno ugrade znanja o specifičnim elementima (slika 6.6). Na primer, možemo definisati šta je to fizički objekat a detalji o konkretnim objektima (televizor, robot,...) mogu se naknadno dobiti. Takava generalna ontologija se naziva gornja ontologija (eng. upper ontology).



Slika 6.6

Postoje dve vrste ontologija, to su ontologije za opšte namene i ontologije za specifične namene. Uopštena ontologija se može primeniti na više ili manje svaki specifičan domen, dok su ontologije za specifične namene specifične za konkretan domen. Većina najboljih VI aplikacija koristi ontologije za specifične namene umesto opštih ontologija. Međutim, postoje projekti koji koriste opšte ontologije, kao što su CYC, DBPedia, TextRunner i OpenMind.

6.10.2 Kategorije i objekti

Organizacija objekata u kategorije je jedna od glavnih koncepata reprezentacija znanja. Iako se iskustva o svetu tumače kao odnosi između objekata, veliki deo rasuđivanja odvija se na nivou kategorija. Na primer, kupac želi da kupi košarkašku loptu, a ne košarkašku loptu KB_{32} . Kategorije se takođe koriste za određivanje osobina objekata preko njegovog pripadanja određenoj kategoriji. Na primer, ako agent na osnovu ulaznih informacija odredi o kom objektu se radi i zna kojoj on kategoriji pripada on može da odredi i ostale osobine objekta na osnovu kategorije. Na primer, ako agent utvrdi na osnovu narandžaste hrapave kore, okruglog oblika da se radi o pomorandži on zaključuje da se taj objekat može (jer pripada određenoj kategoriji) ubaciti u voćnu salatu.

Postoje dva načina da se u logici prvog reda predstave kategorije, to su objekti i predikati. Na primer: $KosarkaskaLopta(k)$ i $KosarkaskaLopta$, $Clan(k, KosarkaskaLopta)$.

Možemo reći i $Podskup(KošarkaškaLopta, Lopta)$ čime smo uveli pojam podkategorije.

Kategorije se koriste da bi se baza znanja mogla bolje organizovati i pojednostaviti korišćenjem koncepta nasleđivanja. Ako kažemo da su svi objekti kategorije *Hrana* jestivi i kažemo da je kategorije *Voće* podkategorija od *Hrana* i *Jabuke* podkategorija od *Voće* možemo zaključiti da su i jabuke jestive. Kažemo da pojedinačne jabuke nasleđuju osobinu jestivosti na osnovu njihove pripadnosti kategoriji *Hrana*.

Odnos podkategorije uređuje kategorije u tzv. taksonomije. Taksonimije se godinama koriste u tehničkim oblastima. Jedna od najpoznatijih taksonomija je iz oblasti bibliotekarstva i naziva se Dewey Decimal system. To je taksonomija svih oblasti znanja. Postoje i taksonomija biljaka i životinja, zanimanja, komercijalnih proizvoda, i sl.

Matematički izrazi i logika prvog reda omogućavaju jednostavno definisanje činjenica o kategorijama. Sledi nekoliko primera.

Objekat pripada kategoriji

$$KL_4 \in KosarkaskaLopta$$

Kategorija je podklasa druge kategorije

$$KosarkaskaLopta \subset Lopta$$

Svi članovi jedne kategorije imaju određene osobine

$$x \in KosarkaskaLopta \Rightarrow Okrugla(x)$$

Članovi kategorije mogu se prepoznati prema nekim osobinama

$$Narandžasta(x) \wedge Okrugla(x) \wedge Prečnik(x) = 9.5" \wedge x \in Lopta \Rightarrow x \in KosarkaskaLopta$$

Kategorija kao celina ima neke osobine

$$Psi \in PripitomljeneVrsteŽivotinja$$

Poslednji primer definiše pripadnost kategorije nekoj drugoj kategoriji.

Pored podkategorije, postoje i drugi važni odnosi između kategorija. Na primer, ako kažemo da su *Mužjaci* i *Ženke* podkategorije klase *Životinje* nismo definisali da mužjak ne može istovremeno biti i ženka. Za dve kategorije kažemo da su disjunktne ako nemaju zajedničkih objekata. Pored disjunkcije, kategorije mogu biti i potpune dekompozicije ukoliko svaki objekat pripada bar nekoj od kategorija. Disjunktne kategorije koje su i potpune dekompozicije nazivaju se particije.

Na primer:

Disjunkcija{Životinje, Biljke},

PotpunaDekompozicija({Amerikanci,Kanađani,Meksikanci}, SevernoAmerikanci),

Particija({Mužjaci, Žanke}, Životinje)

Jedan objekat može biti deo nekog drugog objekta

JeDeo(Bukurešt, Rumunija) (eng. PartOf)

JeDeo(Rumunija, IstočnaEvropa)

JeDeo(IstočnaEvropa, Evropa)

JeDeo(Evropa, Svet)

Ova relacija je tranzitivna i refleksivna

$JeDeo(x, y) \wedge JeDeo(y, z) \Rightarrow JeDeo(x, z)$

$JeDeo(x, x)$

Postoje i posebni način da se predstave mere, događaji, procesi, vremenski intervali , mentalni objekti ali njima se ovde nećemo baviti.

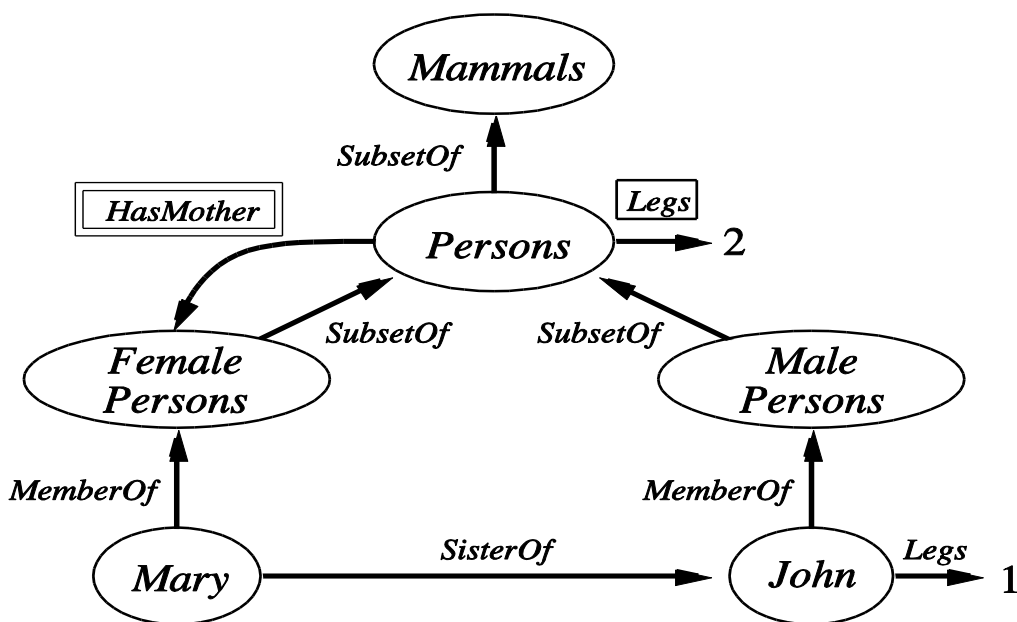
6.3 Sistemi rasuđivanja za kategorije

Kategorije su osnovni gradivni blok velikih baza znanja. Postoje dva sistema za rasuđivanje nad kategorijama, to su semantičke mreže i deskriptivne logike. Semantičke mreže definišu grafičku notaciju kao pomoć vizualizaciji baze znanja i efikasne algoritme za izvođenje zaključaka o osobinama objekata na osnovu pripadnosti kategoriji. Deskriptivne logike obezbeđuju formalni jezik za konstruisanje i kombinovanje definicija kategorija i efikasne algoritme za određivanje odnosa pod i nad kategorije.

Semantičke mreže

U prošlosti je postojala debata između pristalica logike i pristalica semantičkih mreža i kao rezultat te debate izveden je zaključak da su semantičke mreže upravo forma logike. I one imaju koncepte relacije, objekate, kvantifikacije, jedina je razlika što one imaju grafički prijemčiviju notaciju.

Postoji nekoliko varijanti semantičkih mreža ali svaka od njih reprezentuje individualne objekte, kategorije objekata i relacije među njima. Tipična grafička notacija podrazumeva predstavljanje objekata i kategorije u ovalima ili pravougaonicima, a usmerene veze između njih predstavljaju relacije. Primer jedne semantičke mreže dat je na slici 6.7.



Slika 6.7

Semantička mreža prikazana na slici 6.7 sadrži link sa labelom *MemberOf* koji definiše pripadnost objekta kategoriji. Pravougaonicima za dvostrukim okvirom predstavljene su relacije između objekata koji pripadaju kategoriji (kategorije nemaju majke), pravougaonicima sa jednostrukim okvirom osobine objekata koji pripadaju kategoriji.

Notacija semantičke mreže omogućava jednostavno rezonovanje na osnovu nasleđivanja. Na primer, pošto pripada osobama, Meri nasleđuje osobinu da ima dve noge. Znači, da bi odgovorio na pitanje koliko Meri ima nogu algoritam prati linkove sve dok ne dođe do kategorije koja ima definisanu osobinu legs.

Deskriptivna logika

Sintaksa logike prvog reda je dizajnirana da bi se njoj izražavale osobine objekata, a sintaksa deskriptivne logike da bi se opisale definicije i osobine kategorija.

Pojam deskriptivna logika odnosi se na familiju jezika za reprezentaciju znanja. Ona predstavlja logički formalizam za specifikaciju ontologija u okviru Semantičkog weba.

Oznovni zadaci deskriptivne logike su da odgovori na sledeća pitanja:

da li je neka kategorija podkategorija?

da li objekat pripada kategoriji?

da li je definicija kategorije konzistentna?

Primer deskriptivne logike je jezik Classic. Primer izraza u ovom jeziku je:

Bachelor=And(Unmarried, Adult, Male) - definicija kategorije neženja

Opis klase ljudi koji imaju bar tri nezaposlena sina i svi su oženjeni lekarkama, i barem dve kćeri koje su profesorce na departmanu za fiziku ili matematiku:

And(Man, AtLeast(3,Son), AtMost(2, Daughter), All(Son, And(Unemployed, Married, All(Spouse, Doctor))), All(Daughter, And(Professor, Fills(Departmant, Physics, Math))))

Rešavanje problema u deskriptivnoj logici sastoji se od njegovog opisa i provere da li on pripada nekoj od rezultujućih kategorija. Prednost deskriptivne logike u odnosu na logiku prvog reda je brzina rešavanja problema, a nedostatak je što se teži problemi teško opisuju.

Zadaci

6.1 Dati su sledeći simboli:

Zanimanje(o,z) – predikat sa značenjem osoba o ima zanimanje z

Musterija (o1,o2) – predikat sa značenjem osoba o1 je musterija osobi o2

Šef(o1,o2) – predikat sa značenjem osoba o1 je šef osobi o2

Lekar, Hirurg, Advokat, Glumac – konstante koje predstavljaju različita zanimanja

Milica, Marko – konstante koje predstvljaju osobe.

Korišćenjem datih simbola napisati sledeće rečenice korišćenjem logike prvog reda:

- a) Milica je ili advokat ili hirurg.
- b) Marko je glumac, ali ima i drugi posao.
- c) Svi hirurzi su lekari
- d) Marko nema advokata (tj. nije mustiraja ni jednom advokatu)
- e) Milicin šef je advokat.
- f) Postoji advokat čiji svi klijenti (mušterije) su lekari.
- g) Svaki hirurg ima advokata.

6.2. Posmatramo domen geografskih regiona i obojenu mapu. Rapolažemo funkcijom *BojaMapa(x)* koja predstavlja boju države x na mapi i predikatima $x=y$, *Deo(x,y)* koji označava da je region x deo regiona y, *Susedni(x,y)* koji označava da su regioni x i y sysedni, i *Država(x)* koji označava da je region x država. Imamo i konstante koje označavaju regione (Evropa, JuznaAmerika). Date su dve rečenice na srpskom i za svaku od njih 4 moguće rečenice u logici prvog reda. Pored svake od 4 logičke rečenice treba upisati jedan od sledećih brojeva:

1 - ako logička rečenica ima isto značenje kao rečenica na srpskom

2 - ako je logička rečenica sintaksno neispravna

3 - ako je rečenica sintaksno ispravna ali nema isto značenje kao rečenica na srpskom

A) Nijedan region u Južnoj Americi se ne graniči sa regionom u Evropi.

$$\neg[\exists c, d \text{ Deo}(c, \text{JuznaAmerika}) \wedge \text{Deo}(d, \text{Evropa}) \wedge \text{Susedni}(c, d)]$$

$$\forall c, d [\text{Deo}(c, \text{JuznaAmerika}) \wedge \text{Deo}(d, \text{Evropa}) \Rightarrow \neg \text{Susedni}(c, d)]$$

$$\neg \forall c \text{ Deo}(c, \text{JuznaAmerika}) \Rightarrow \exists d \text{ Deo}(d, \text{Evropa}) \wedge \neg \text{Susedni}(c, d)$$

$$\forall c \text{ Deo}(c, \text{JuznaAmerika}) \Rightarrow \forall d \text{ Deo}(d, \text{Evropa}) \Rightarrow \neg \text{Susedni}(c, d)$$

B) Dve susedne države nemaju istu boju na mapi.

$$\forall x, y \neg \text{Država}(x) \vee \neg \text{Država}(y) \vee x = y \vee \neg \text{Susedni}(x, y) \vee \neg (\text{BojaMapa}(x) = \text{BojaMapa}(y))$$

$$\forall x, y \text{ Država}(x) \wedge \text{Država}(y) \wedge \neg(x = y) \wedge \text{Susedni}(x, y) \Rightarrow \neg (\text{BojaMapa}(x) = \text{BojaMapa}(y))$$

$$\forall x, y \text{ Država}(x) \wedge \text{Država}(y) \wedge \neg(x = y) \wedge \text{Susedni}(x, y) \wedge \neg (\text{BojaMapa}(x) = \text{BojaMapa}(y))$$

$$\forall x, y (\text{Država}(x) \wedge \text{Država}(y) \wedge \neg(x = y) \wedge \text{Susedni}(x, y)) \Rightarrow \text{BojaMapa}(x \neq y)$$

6.3. Da li sledeće rečenice na srpskom i u logici prvog reda odgovaraju jedna drugoj po značenju.

a) "Ne postoje dve osobe koje imaju isti JMBG"

$$\neg \exists x, y, n \text{ Osoba}(x) \wedge \text{Osoba}(y) \Rightarrow \text{ImaJmbg}(x, n) \wedge \text{ImaJmbg}(y, n)$$

b) "Petrov JMBG je isti kao Milanov"

$$\exists n \text{ ImaJmbg}(\text{Petar}, n) \wedge \text{ImaJmbg}(\text{Milan}, n)$$

c) "Svaka osoba ima JMBG koji se sastoji od 11 cifara"

$$\forall x, n \text{ Osoba}(x) \Rightarrow \text{ImaJmbg}(x, n) \wedge \text{Cifre}(n, 11)$$

Preformulisati rečenice predikatskog računa (ispraviti one koje su netačne) korišćenjem funkcije Jmbg(x) umesto predikata ImaJmbg(x,n)

6.4. Data je baza znanja

$$P(F(x)) \Rightarrow P(x)$$

$$Q(x) \Rightarrow P(F(x))$$

$$P(A)$$

$$Q(B)$$

a) Da li se algoritmom ulančavanja unapred može zaključiti Q(A)?

b) Da li se algoritmom ulančavanja unpred može zaključiti P(B)?

c) Da li algoritam ulančavanja unazad vraća tačno za upit $P(B)$?

d) Da li algoritam ulančavanja unazad vraća tačno za upit $P(A)$?

Poglavlje 7

Mere nesigurnosti u Veštačkoj inteligenciji

U velikom broju slučajeva agent mora da se "nosi" sa nekom nesigurnošću. Razlozi za to mogu biti da se okruženje ne može potpuno sagledati ili da se radi o nedeterminističkom okruženju ili oba istovremeno. Moguće je da agent ne može biti siguran u kom stanju se nalazi ili gde će završiti nakon neke sekvence akcija.

Agenti za rešavanje problema pretraživanjem i logički agenti su savladavali nesigurnost predstavljanjem svih mogućih stanja u kojima se agent može naći i generisanjem plana akcija na osnovu razmatanja svih mogućih akcija. Ovaj pristup ima određene nedostatke, a neki od njih su:

- kada interpretira parcijalne informacije sa senzora, logički agent mora da uzme u obzir sve logički moguće varijante zapažanja, bez obzira koliko je svaka od njih verovatna, što vodi do jako velikih i kompleksnih reprezentacija stanja
- ponekad se ne može napraviti plan koji sigurno vodi do cilja, a agent ipak mora da izvrši neku akciju i bilo bi korisno da postoji neki sistem da se ocene osobine mogućih rešenja

Posmatrajmo automatskog taksi vozača čiji je cilj da odveze putnika na aerodrom tako da on ne zakasni na let. Agent pravi plan A_{90} koji podrazumeva polazak od kuće 90 minuta pre polaska aviona uz vožnju razumnom brzinom. Iako je aerodrom udaljen 6 km, taksi agent ne može sa sigurnošću da tvrdi da će ga plan A_{90} sigurno dovesti na aerodrom na vreme. Umesto toga možemo reći da će ovaj plan biti uspešan ako se auto ne pokvari ili ako ne nestane goriva i ako se ne desi saobraćajna nezgoda i ako se na mostu nije desila saobraćajna nesreća, i ako avion ne krene ranije i ako meteorit ne udari auto,... Iako ne možemo zasigurno znati da li će nas ovaj plan odvesti do cilja on ipak predstavlja razumnu stvar koju treba uraditi. Šta znači razumna stvar? To znači da od svih mogućih planova očekivano je da plan A_{90} ima maksimalnu meru učinka, koja uključuje dolazak na aerodrom na vreme i izbeći dugo čekanje na aerodromu i izbeći kazne za prebrzu vožnju. Drugi planovi kao što su A_{180} imaju veću verovatnoću da će agent stići na vreme, ali takođe i da će duže čekati na aerodromu.

Možemo zaključiti da racionalna odluka zavisi od dve stvari, značaj različitih ciljeva i verovatnoća da će se oni dostići i do koje mere.

Izražavanje nesigurnosti

Posmatrajmo sada problem predstavljanja nesigurnih informacija: dijagnoza za zubobolju. Sve vrste dijagnoza, u medicini, popravci automobila i slično, skoro uvek uključuju nesigurnost.

Ako bismo hteli da pravila za zubarske dijagnoze predstavimo logikom prvog reda imali bismo sledeća pravila (u zagradi su dati problemi izabranog načina reprezentacije):

Zubobolja \Rightarrow Karijes (ne mora da znači da pacijent ima karijes ako ga boli zub)

Zubobolja \Rightarrow Karijes \vee BolesneDesni \vee ... (velika lista mogućih uzroka)

Karijes \Rightarrow Zubobolja (ne mora da znači da će pacijenta boleti zub ako ima karijes)

Korišćenje logike za domen kao što su medicinske dijagnoze je nedovoljno iz tri razloga:

- ogroman broj uzroka i posledica
- nedovoljno teoretsko znanje jer medicina nema kompletnu teoriju o problemu
- nedovoljno praktično znanje jer nisu izvršena sva ispitivanja.

Odnos između zubobolje i karijesa i nije logička posledica u bilo kom smeru. Ovo je tipično za oblast medicine, ali i za mnoge druge oblasti: pravo, ekonomija, dizajn, popravka automobila, baštovanstvo. U ovim oblastima agent ima samo određeni stepen verovanja da je nešto tačno. Osnovno oruđe za obradu "stepena verovanja" je teorija verovatnoće.

Dok logički agent veruje da je svaki iskaz tačan ili netačan (1 ili 0) agent koji koristi verovatnoću može numerički da izrazi verovanje da je nešto tačno (vrednostima između 0 i 1). Na primer, možemo reći da u 80% slučajeva zubobolje radi se o karijesu, odnosno verovatnoća da neko ko ima zubobolju ima karijes je 0.8.

7.1 Teorija verovatnoće

Notacija:

$P(A)$ - verovatnoća da je A istinito.

Ako posmatramo slučaj bacanja dve kockice (za jamb), možemo imati 36 mogućih ishoda, to su (1,1), (1,2), ..., (6,6). Ove pojedinačne ishodi se u teoriji verovatnoće nazivaju elementarni događaji i obeležavaju se sa ω . Skup svih elementarnih događaja naziva se siguran događaj i obeležava se sa Ω . Proizvoljni podskup od Ω naziva se događaj. Osnovni aksiom teorije verovatnoće kaže da svaki elementarni događaj ima verovatnoću veću ili jednaku od 0 i manju ili jednaku od 1 i da je zbir verovatnoća svih elementarnih događaja iz skupa Ω jednak 1.

$$0 \leq P(\omega) \leq 1$$

$$\sum_{\omega \in \Omega} P(\omega) = 1$$

Verovatnoća da u bacanju dve kocke padne jedna kombinacija je $1/36$ i ovde se radi o verovatnoći elementarnog događaja.

Tvrdnje i upiti koji uključuju verovatnoću često se ne odnose samo na verovatnoću nekog elementarnog događaja, već na verovatnoću događaja (skupa elementarnih događaja). Na primer, koja je verovatnoća

da bacanje dve kocke daje zbir 11. U ovom slučaju imamo verovatnoću događaja (ϕ) koja se računa na sledeći način:

$$P(\phi) = \sum_{\omega \in \phi} P(\omega)$$

Odnosno u slučaju bacanja kockica, $P(\text{zbir}=11)=P((5,6))+P((6,5))=1/36+1/36=1/18$.

$$P(\neg A) = 1 - P(A)$$

Verovatnoća izražena sa $P(A)$ se naziva bezuslovna verovatnoća, ili apriori verovatnoća i predstavlja verovatnoću da je nešto tačno u odsustvu bilo kojih dodatnih informacija. $P(A \vee B)$ je bezuslovna verovatnoća da je A ili B istinito i važi:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Pored bezuslovne verovatnoće postoji i uslovna verovatnoća koja se označava sa $P(A|B)$ koja definiše verovatnoću da je tačno A uz uslov da je ispunjeno B. Na primer, izraz $P(\text{karijes}|\text{zubobolja}) = 0.6$ znači da ako pacijent ima zubobolju (i nemamo dodatnih informacija) onda je verovatnoća da on ima karijes 0.6.

Uslovna verovatnoća se definiše preko bezuslovne na sledeći način:

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Iz ovoga se može zaključiti da važi:

$$P(A \wedge B) = P(A|B)P(B)$$

$$P(A|B)P(B) = P(B|A)P(A).$$

Kažemo da A ne zavisi od B ako važi $P(A|B)=P(A)$. Ako važi da i B ne zavisi od A $P(B|A)=P(B)$, sledi $P(A \wedge B)=P(A)P(B)$. Primer: $P(\text{zubobolja}|\text{oblačno}) = P(\text{zubobolja})$

Bajesovo pravilo

Na osnovu $P(A|B)P(B) = P(B|A)P(A)$ možemo izvesti:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Ovo pravilo se naziva Bajesovo pravilo i ono predstavlja osnovu za skoro sve moderne VI sisteme za rezonovanje u prisustvu nesigurnosti. Ono nam omogućava da izračunamo uslovnu verovatnoću čak i ako ne znamo $P(A \wedge B)$.

Primer: Bubuljice i boginje

Verovatnoća da boginje daju bubuljice je 0.5.

Verovatnoća da osoba ima boginje je 0.1.

Verovatnoća da osoba ima bubuljice je 0.2.

Koja je verovatnoća da osoba koja ima bubuljice ima boginje?

$$P(\text{Bubuljice} \mid \text{Boginje}) = 0.5$$

$$P(\text{Boginje}) = 0.1$$

$$P(\text{Bubuljice}) = 0.2$$

$$P(\text{Boginje} \mid \text{Bubuljice}) = \frac{P(\text{Bubuljice} \mid \text{Boginje})P(\text{Boginje})}{P(\text{Bubuljice})}$$

$$P(\text{Boginje} \mid \text{Bubuljice}) = \frac{(0.5)(0.1)}{0.2} = 0.25$$

Zašto je Bajesovo pravilo važno?

U implementaciji ekspertnih sistema često raspoložemo klauzalnim znanjima, a želimo da izvedemo rasuđivanje na osnovu nekih posledica.

$$P(\text{uzrok} \mid \text{posledica}) = \frac{P(\text{posledica} \mid \text{uzrok})P(\text{posledica})}{P(\text{uzrok})}$$

Primer: U medicini imamo znanja oblika $P(\text{simptom} \mid \text{bolest})$, a dijagnozu uspostavljamo na sledeći način $P(\text{bolest} \mid \text{simptom})$.

Relativne verovatnoće

Ako znamo da je $P(A \mid B_1) = P(A \mid B_2)$, tada se iz Bajesovog pravila može izvesti:

$$\frac{P(B_1 \mid A)}{P(B_2 \mid A)} = \frac{P(B_1)}{P(B_2)}$$

Kako nam znanje o B_1 kaže o A kao i znanje o B_2 , znanje o A ne menja relativnu verovatnoću za B_1 i B_2 .

Primer: semafor

Verovatnoća za semafor T:

- Zeleno je 0.45
- Žuto je 0.1
- Crveno je 0.45

Svako ko pređe na crveno dobija kaznu (uvek) i niko drugi ne dobija kaznu.

Prešli smo baz kazne – koja je verovatnoća da je svetlo bilo žuto? Zeleno? Crveno?

$$P(\text{NeKazna} | \text{Crveno}) = 0 \text{ i } P(\text{Crveno} | \text{NeKazna}) = 0$$

$$P(\text{NeKazna} | \text{Zeleno}) = P(\text{NeKazna} | \text{Žuto}) = 1, \text{ dakle važi}$$

$$\frac{P(\text{Žuto} | \text{NeKazna})}{P(\text{Zeleno} | \text{NeKazna})} = \frac{P(\text{Žuto})}{P(\text{Zeleno})} = \frac{0.1}{0.45}$$

A znamo i da važi:

$$P(\text{Žuto} | \text{NeKazna}) + P(\text{Zeleno} | \text{NeKazna}) + P(\text{Crveno} | \text{NeKazna}) = P(\text{Žuto} | \text{NeKazna}) + P(\text{Zeleno} | \text{NeKazna}) = 1.0$$

$$P(\text{Žuto} | \text{Ne-Kazna}) = (0.1/0.55) \sim 0.18$$

$$P(\text{Zeleno} | \text{Ne-Kazna}) = (0.45/0.55) \sim 0.8$$

Primer: šibicarenje

Zrno je sakriveno ispod jedne od tri kutije šibice, zadatak vam je da odaberete pravu kutiju.

Kada izaberete kutiju, operater obrne kutiju koju niste odabrali i ispod koje nije zrno. On vam dozvoljava da promenite izbor.

Da li treba da promenite Vaš izbor?

Bajesovska analiza

Označimo kutije šibice sa A, B, C. Svaki od iskaza A, B, C je istinit ako je zrno ispod odgovarajuće kutije.

Pretpostavimo da ste odabrali kutiju A. Želimo da izračunamo verovatnoću da ćete dobiti ako ostanete pri istom izboru i verovatnoću da ćete dobiti ako promeniti izbor.

Ako ostanete pri istom izboru verovatnoća je

$$P(A | \neg B \vee \neg C) = \frac{P(\neg B \vee \neg C | A)P(A)}{P(\neg B \vee \neg C)}$$

$P(\neg B \vee \neg C | A)$ je 1.0: ako znamo da je zrno ispod A, onda znamo da nije ispod B ili C.

$$P(A) = 1/3$$

$P(\neg B \vee \neg C)$ je 1.0, jer ne može da se nalazi i ispod B i ispod C.

$P(A | \neg B \vee \neg C) = 1/3$. Ovo je za očekivanje, jer prevrtanje prazne kutije ne menja verovatnoću originalanog izbora.

Ako promenite izbor verovatnoća je:

$$P(\neg A | \neg B \vee \neg C) = \frac{P(\neg B \vee \neg C | \neg A)P(\neg A)}{P(\neg B \vee \neg C)}$$

$P(\neg B \vee \neg C | \neg A)$ je 1, jer zrno ne može da se nalazi i ispod B i ispod C.

$$P(\neg A) = 2/3$$

$P(\neg B \vee \neg C)$ je 1.0, jer zrno ne može da se nalazi i ispod B i ispod C.

$$P(\neg A | \neg B \vee \neg C) = 2/3.$$

Trebalo bi da promenite izbor!

Intuicija:

Ako ostajete pri prvom izboru, pobeđujete u 1/3 slučajeva.

Ako promenite izbor tada pobeđujete ukoliko vaš originalni izbor nije bio dobar. To se dešava u 2/3 slučajeva.

7.2 Bajesova mreža

Bajesova mreža predstavlja grafičku notaciju za predstavljanje uslovnih verovatnoća. Bajsova mreža ima ulogu sličnu logici prvog reda u svetu u kom imamo sigurne iskaze.

Sintaksu bajesove mreže čini skup čvorova (za svaku promenljivu po jedan). Bajesova mreža je usmereni, aciklični graf, a grana se tumači kao uticaj čvora roditelja na čvor dete. Za svaki čvor se navodi uslovna verovatnoća $P(X_i | \text{Parent}(X_i))$. Za čvorove koji nemaju roditelja navodi se apriori verovatnoća.

Posmatrajmo sledeći problem:

Imamo novi alarm na kući koji je prilično pouzdan u detektovanju provalnika, ali ponekad se uključi i u slučaju zemljotresa.

Takođe imamo i dva suseda John i Mary koji su obećali da će nas pozvati ako čuju alarm.

John će sigurno pozvati ako se uključi alarm, ali se može desiti da zvono na vratima pomeša sa alarmom i da i tada pozove

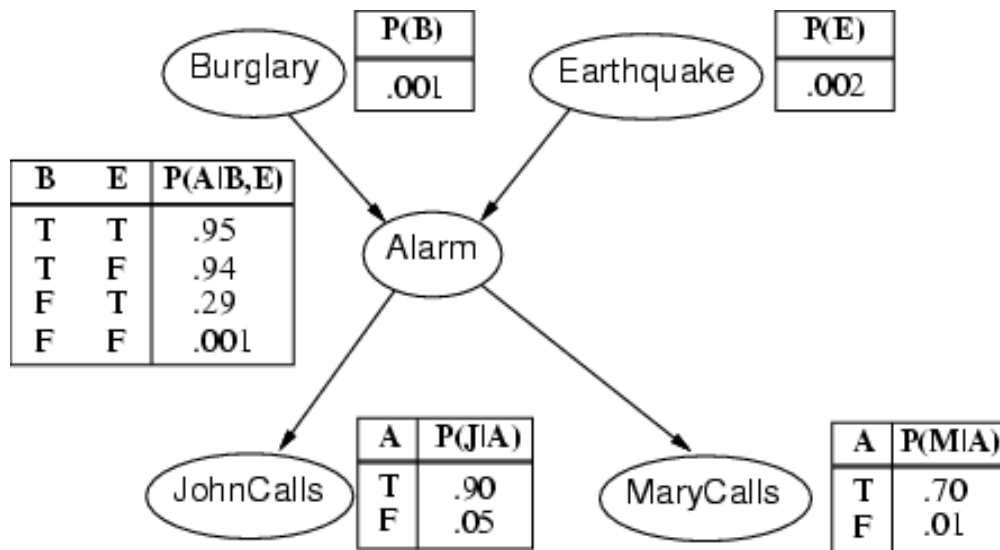
Mary voli glasnu muziku i često ne čuje alarm.

Ako jedan od komšija pozove, koja je verovatnoća da se radi o provalniku?

Na slici 7.1 prikazana je Bajesova mreža za ovaj problem. Struktura mreže pokazuje da provala i zemljotres ne utiču na to da li će komšije da pozovu. Da li će Meri ili John zvati zavisi samo od toga da li će se oglasiti alarm. Dakle, u kreiranje mreže uključili smo pretpostavku da komšije ne čuju provalu i da ne primećuju mali zemljotres i ne izlaze na prozor da provere kada čuju alarm.

Pored topologije mreže na slici 6.1 prikazane su i tabele uslovnih verovatnoća (conditional probability table - CPT). Ovakve tabele se mogu koristiti za diskretne promenljive, kontinualne promenljive se drugačije prikazuju. Svaka vrsta u tabeli uslovnih verovatnoća sadrži uslovnu verovatnoću za jedan uslovni slučaj (conditioning case). Uslovni slučaj je jedna kombinacija vrednosti za roditeljske čvorove. Čvorovi koji nemaju roditelje sadrže vrednost bezuslovnih verovatnoća.

Treba primetiti da mreža ne sadrži čvorove za slušanje muzike ili zvona na vratima koje mogu da zbune Johna. Svi ti faktori su sumirani u verovatnoći koja povezuje alarm za JohnCalls i MaryCalls. Verovatnoće sumiraju potencijalno beskonačan skup okolnosti u kojima se nešto može dogoditi.



Slika 7.1

Semantika Bajesove mreže

Hoćemo da izračunamo koja je verovatnoća da se oglašio alarm a da nije bilo ni provale ni zemljotresa i da su obe komšije nazvale.

$$P(j, m, a, \neg b, \neg e) = ?$$

Važi pravilo:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i \mid \text{parents}(X_i))$$

gde $\text{parents}(X_i)$ označava vrednost roditelja od X_i ($\text{Parents}(X_i)$) koja se pojavljuje u $x_1 \dots x_n$.

Ovo pravilo važi kada posmatramo vrednosti svih promenljivih na Bejesovoj mreži što se naziva kompletna zajednička distribucija verovatnoća (eng. full joint probability distribution). Na osnovu ove distribucije verovatnoća može se dati odgovor na bilo koje pitanje nad domenom za koje znamo vrednosti svih promenljivih.

Dakle:

$$P(j, m, a, \neg b, \neg e) = P(j|a)P(m|a)P(a|\neg b \wedge \neg e)P(\neg b)P(\neg e) = 0.90 * 0.70 * 0.001 * 0.999 * 0.998 = 0.000628$$

Zadaci:

7.1 Data je vreća sa tri novčića, a, b i c. Verovatnoća da bacanjem novčinja dobijemo glavu je redom 0.3, 0.6 i 0.75. Svaki od novčića će biti izabran iz vreće sa jednakom verovatnoćom. Nakon izbora jednog novčića on se baca tri puta i generiše tri ishoda X_1 , X_2 i X_3 .

a) Nacrtati Bajesovu mrežu za ovaj problem

b) Izračunati koji novčić je najverovatnije izabran iz vreće ako je rezultat bacanja bio dve glave i pismo.

7.2 Posmatramo sledeći problem:

Dolazimo kući i želimo da znamo, da li ima nekoga kod kuće pre nego što otvorimo vrata. Svetlo ispred kuće može biti upaljeno. Supruga ga često upali kad ode negde, a upali ga i kada očekuje goste. Imamo psa koga vodimo iza kuće ako nema nikoga kod kuće. Vodimo ga iza kuće i kada ima problema sa stomakom. Ako je pas iza kuće, možemo ga čuti kako laje, ali se taj lavež može zameniti i sa lavežom drugih pasa.

a) Nacrtati Bajesovu mrežu za ovaj problem.

b) Dodeliti tablicu verovatnoća čvorovima

Poglavlje 8

Odlučivanje u sistemima VI

U svojoj osnovnoj formi, teorija odlučivanja se bavi izborom akcije iz nekog skupa mogućih akcija na osnovu njihovih neposrednih ishoda. U rešavanju problema pretraživanjem koristili smo notaciju $\text{Result}(s_0, a)$ za stanje koje je u determinističkom okruženju bilo rezultat primene akcije a na stanje s_0 .

U ovom odeljku razmatramo nedeterminističko okruženje koje se ne može potpuno sagledati. Pošto agent ne mora da zna kakvo je trenutno stanje (jer se okruženje ne može potpuno sagledati) rezultat predstavljamo sa $\text{Result}(a)$ kao proizvoljnu promenljivu čije su vrednosti stanja koja su mogući ishodi akcije a . Verovatnoća ishoda s' , uz zapažanja iz okruženja e označavamo na sledeći način:

$$P(\text{Result}(a) = s' | a, e),$$

gde a sa desne strane linije označava događaj koji je izazvan akcijom a .

Agentove preferencije se izražavaju funkcijom korisnosti (eng. utility function), $U(s)$, numeričkom vrednošću koja izražava poželjnost stanja. Očekivana korisnost (eng. expected utility) akcije uz zapažanje e označava se sa $EU(a|e)$ i izračunava na sledeći način:

$$EU(a|e) = \sum_{s'} P(\text{Result}(a) = s' | a, e) U(s')$$

Princip maksimalne očekivane korisnosti (eng. maximum expected utility - MEU) kaže da racionalni agent treba da izabere akciju koja maksimizira njegovu očekivanu korisnost:

$$action = \arg \max_a EU(a|e)$$

8.1 Funkcija korisnosti

$U(A)$ – funkcija korisnosti (utility function) je broj koji označava poželjnost nekog ishoda. Važi sledeće:

$$U(A) > U(B) \Leftrightarrow A \succ B \text{ (preferiramo ishod A u odnosu na ishod B)}$$

$$U(A) = U(B) \Leftrightarrow A \sim B \text{ (indiferentni smo (sve jedno nam je) u odnosu na ishode A i B)}$$

Uvodimo sledeće oznake

$$U(S) = u_{\top} - \text{najbolji mogući ishod}$$

$$U(S) = u_{\perp} - \text{najgori mogući ishod}$$

Normalizovana korisnost koristi skalu $u_{\perp} = 0$ i $u_{\top} = 1$

Koreni teorije korisnosti su u ekonomiji koja ima poznatu funkciju korisnosti a to je novac. Obično je slučaj da agent preferira više novca, međutim to ne znači da je količina novca funkcija korisnosti jer on ne govori ništa o verovatnoćama određenih ishoda koje uključuju novac.

Pretpostavimo da ste pobedili na nekom kvizu i da vam se ponudi izbor, odmah osvajate 1 milion dolara ili bacamo novčić, ako padne glava odlazite bez ičega, a ako padne pismo dobijate 2,5 miliona dolara. Većina ljudi bi odustala od kocke. Da li je ovo racionalno ponašanje?

Pretpostavimo da je ista verovatnoća da padne glava i pismo i izračunamo funkciju korisnosti koja se u ekonomiji zove očekivana novčana vrednost (expected monetary value - EMV).

$$EMV(\text{PrihvatiKocku}) = 1/2 \cdot 0 + 1/2 \cdot 2\,500\,000 = 1\,250\,000.$$

Što znači da je očekivana novčana vrednost prihvatanja kocke veća od vrednosti koju ćete imati ako ne prihvatite kocku, ali to ne znači da je bolje prihvatiti. Treba razmotriti drugačiju funkciju korisnosti.

Sa S_n ćemo obeležiti stanje u kojem posedujemo n dolara. Pretpostavimo da je trenutno stanje S_k . Sada je očekivana novčana vrednost

$$EU(\text{PrihvatiKocku}) = 1/2 \cdot U(S_k) + 1/2 \cdot U(S_{k+2\,500\,000})$$

$$EU(\text{NeKocka}) = U(S_{k+1\,000\,000})$$

Da bismo odredili šta da radimo treba odrediti korisnost ishoda. Definisaćemo korisnost tako da prvi milion ocenjujemo bolje od narednih. Na primer, za trenutno stanje S_k dodelimo vrednost 5, za $S_{k+2\,500\,000}$ dodelimo vrednost 9, a za $S_{k+1\,000\,000}$ dodelimo 8. U ovakvom slučaju dobijamo da je racionalna odluka da odustanemo. Sa druge strane, za milionere bi funkcija korisnosti bila lokalno linearna u odnosu na koji milion više i on bi prihvatio kocku.

Ljudska iracionalnost (strana 620)

8.2 Funkcija korisnosti sa više atributa

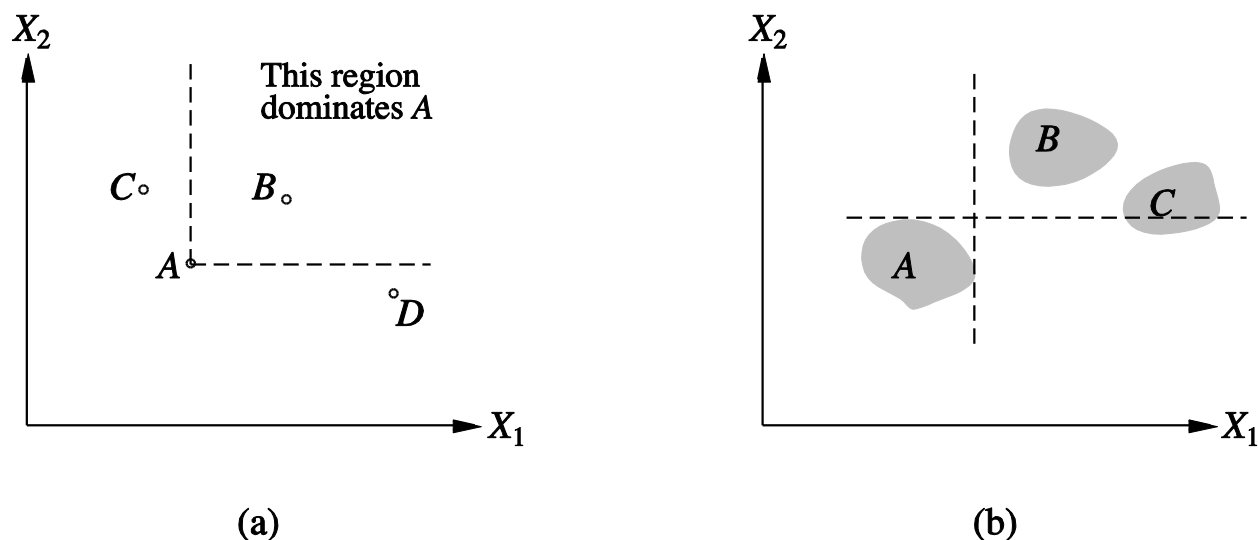
Odlučivanje često uključuje više faktora. Na primer, odluka o dozvoljenoj emisiji zračenja iz nuklearne elektrane mora da uzme u obzir prevenciju posledica na zdravlje stanovništva, na privredu okoline (naročito poljoprivredu) sa jedne strane i ekonomsku dobit od energije sa druge strane. Drugi primer je izbor lokacije za izgradnju aerodroma koja zahteva razmatranje cene zemljišta, udaljenost od naseljenih mesta (ni premala ni prevelika), uticaj buke aviona, sigurnost letova u odnosu na topologiju i vremenske uslove.

Ovo su problemi u kojima se ishodi karakterišu kao vrednosti dva ili više atributa.

Atribute ćemo označavati sa $\mathbf{X} = X_1, \dots, X_n$, a njihove vrednosti sa $\mathbf{x} = x_1, \dots, x_n$ gde je svako x_i je neka numerička vrednost pri čemu se veće vrednosti tumače kao bolje. Na primer, ako imamo atribut *ManjeBuke* u problemu sa aerodromom, veća vrednost će značiti bolje rešenje, odnosno manje buke.

Pretpostavimo da lokacija za aerodrom S_1 košta manje, emituje manje buke, i sigurnija je od lokacije S_2 . Svako će izabrati lokaciju S_1 bez dvoumljenja. Tada kažemo da postoji striktna dominacija S_1 nad S_2 . Uopšteno, ako jedna opcija ima veće vrednosti svih atributa u odnosu na drugu opciju, onda prva opcija dominira drugom.

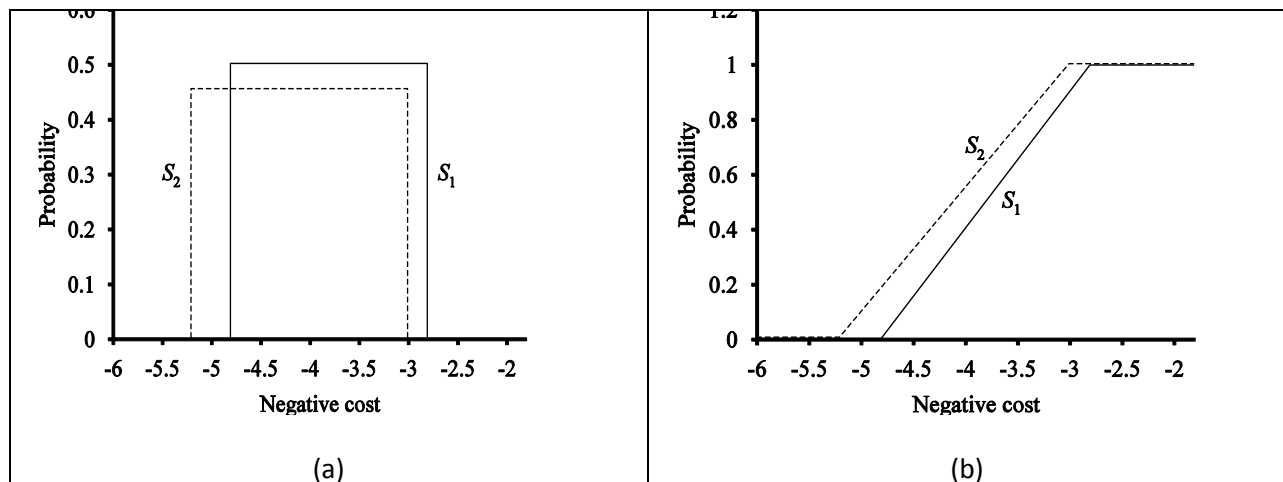
Na slici 8.1 (a) prikazana je dominacija u slučaju sa sva atributa. Ovakav opis važi za determinističke slučajeve u kojima su vrednosti atributa zasigurno tačne. Šta je sa slučajevima kada ne znamo tačne vrednosti atributa već ih procenjujemo u nekom intervalu i sa nekim verovatnoćama. Ovakav slučaj je ilustrovan na slici 8.1 (b). Značenje dominacije u ovom slučaju je da su svi mogući ishodi atributa S_1 veći od svih mogućih ishoda S_2 .



Slika 8.1

U realnim problemima se više koristi stohastička dominacija, koju je najlakše objasniti u kontekstu jednog atributa. Na primer, ako pretpostavimo da je cena izgradnje aerodroma na mestu S_1 između 2.8 miliona dolara i 4.8 miliona dolara, a cena izgradnje na mestu S_2 je između 3 miliona i 5.2 miliona. Slika 8.2 (a) pokazuje ovaj slučaj. Ako uzmemo u obzir da se funkcija korisnosti smanjuje sa povećanjem troškova, možemo reći da S_1 stohastički dominira nad S_2 (i izbor S_2 se može odbaciti). Opisano rezonovanje se ne može primeniti ako imamo tačne vrednosti, na primer ako znamo da je cena S_1 tačno 3.8 miliona.

Precizna stohastička dominacija se može izraziti preko kumulativnih distribucija (slika 8.2 (b)) koje mere verovatnoću da je cena manja ili jednaka nekoj vrednosti. Ako je linija kumulativne distribucije za S_1 uvek desno od linije kumulativne distribucije za S_2 onda zaključujemo da je stohastički posmatrano, S_1 jeftinije od S_2 .



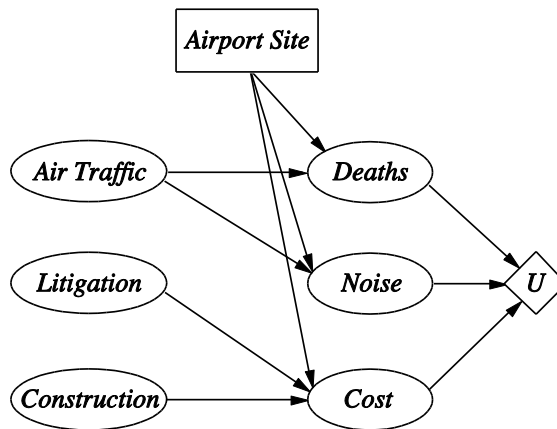
Slika 8.2

8.4 Mreže odlučivanja

Mreže odlučivanja su notacija za generalni mehanizam donošenja racionalnih odluka. One predstavljaju kombinaciju Bajesovih mreža za dodatnim tipovima čvorova za akcije i korisnost. Ilustriramo ih na primeru lokacije aerodroma.

U najopštijem obliku mreže odlučivanja predstavljaju informacije o trenutnom stanju agenta, njegovim mogućim akcijama, stanju koje će biti rezultat izvršavanja akcija i korisnost stanja. Na slici 8.3 prikazana je mreža odlučivanja za problem lokacije aerodroma. Ovaj primer mreže ilustruje upotrebu tri osnovna tipa čvorova:

- Čvorovi promenljivih – chance nodes (ovalni) koji predstavljaju promenljive, slično kao kod Bajesovih mreža. Agent može imati nesigurne informacije o ceni izgradnje, nivou vazdušnog saobraćaja i mogućim slučajevima parničnih postupaka. Promenljive *Deaths*, *Noise* i *Cost* zavise od lokacije aerodroma. Za svaki čvor promenljivih vezuje se uslovna verovatnoća na osnovu verovatnoće roditelja (kao kod Bajesovih mreža). U mrežama odlučivanja, roditeljski čvor može da bude čvor promenljivih ili čvor odluke.
- Čvorovi odluke – decision nodes (pravougaonici) predstavljaju tačke gde onaj koji donosi odluku ima izbor akcija. U ovom slučaju to je *AirportSite* koji označava izbor različitih lokacija koji su u optičaju. Izbor lokacije će uticati na cenu, sigurnost i buku.
- Čvorovi korisnosti – utility nodes (rombovi) predstavljaju funkciju korisnosti agenta. Čvorovi korisnosti imaju za roditelje sve promenljive koje utiču na korisnost. Za ove čvorove se vezuje opis agentove korisnosti kao funkcija čije su promenljive roditelji čvora. Taj opis može biti tabelarni prikaz funkcije, ili može biti neka linearna funkcija nad vrednostima atributa.



Slika 8.3

Izbor akcije vrši se na osnovu procene mreže odlučivanja za svaku vrednost čvora odluke. Kada se čvoru odluke dodeli neka vrednost on dobija ulogu čvora promenljive.

Procena mreže odlučivanja vrši se na sledeći način, za svaku moguću vrednost čvora odlučivanja, postavlja se vrednost čvora, zatim se izračunavaju sve uslovne verovatnoće za roditelje čvora korisnosti, zatim se izračunava rezultujuća korisnost. Rezultat je akcija sa najvećom vrednošću funkcije korisnosti.

8.5 Ekspertni sistemi zasnovani na teoriji odlučivanja

Oblast analiza odlučivanja, koja se razvijala od 1950. do 1960. izučava primenu teorije odlučivanja na rešavanje realnih problema odlučivanja i ima ulogu pomoći pri donošenju racionalnih odluka u sistemima u kojima imamo velike uloge, kao što su biznis, pravo, vojne strategije, medicinske dijagnoze, javno zdravlje i upravljanje resursima.

Proces kreiranja ekspertnih sistema zasnovanih na odlučivanju obuhvata detaljnu analizu mogućih akcija i njihovih ishoda, kao i preferencije u odnosu na te ishode. Rana istraživanje u oblasti razvoja ekspertnih sistema bavila su se kreiranjem odgovora na postavljena pitanja. Ovi sistemi jesu davali neke preporuke o akcijama, ali se nisu bavili eksplicitnim reprezentacijama ishoda i preferencija, već su se zasnivali na pravlima oblika uslov-akcija. Bajesove mreže omogućile su razvoj sistema sa korektnim rezonovanjem u prisustvu nesigurnosti. Uvođenje mreža odlučivanja znači da se mogu razvijati ekspertni sistemi koji preporučuju optimalne odluke koje odražavaju preferencije agenta.

U nastavku će biti opisan proces inženjerstva znanja za ekspertne sisteme koji uključuju teoriju odlučivanja na primeru medicinskog problema određivanja tretmana za pacijente sa urođenom srčanom manom.

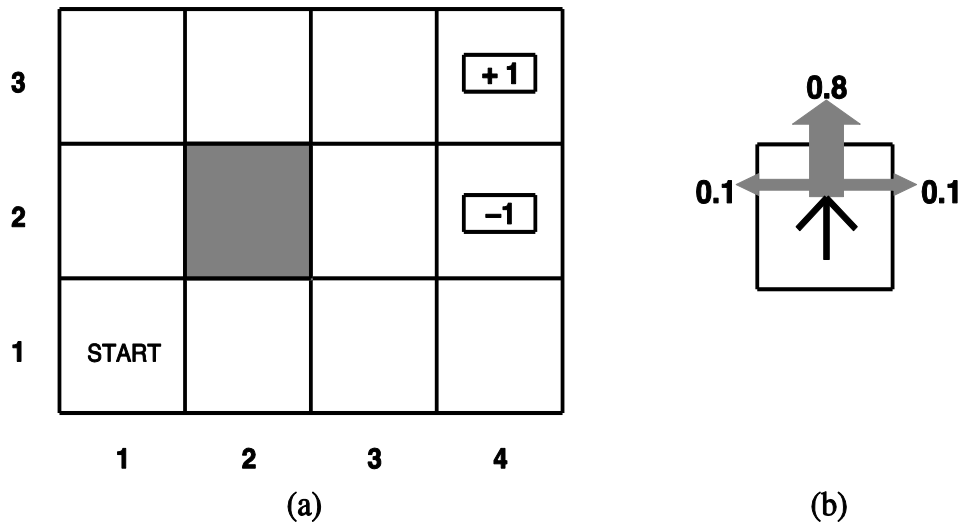
Oko 0.8% ljudi se rađa sa nekom srčanom anomalijom koja se može izlečiti operacijom ili lekovima. Problem je odlučiti koji tretman dati pacijentu i kada (ako se radi o malom detetu rizik je veći, ali se ne sme ni predugo čekati). Tim koji bi radio na izradi ekspertnog sistema za ovaj problem trebalo bi da se sastoji od bar jednog eksperta iz ove oblasti (kardiologa) i jednog projektanta znanje (eng. knowledge engineer). Proces izrade ovakvog sistema sastojao bi se iz sledećih delova:

1. Kreiranje modela uzroka i posledica. Određivanje mogućih simptoma, poremećaja, mogućih lečenja i ishoda. Zatim treba nacrtati veze između ovih elemenata koje određuju koji poremećaj prouzrokuje koje simptome i koji tip lečenja može da pomogne u slučaju kojih poremećaja. Nešto od ovoga će nam reći ekspert, a neke podatke moramo uzimati iz stručne literature.
2. Pojednostavljenje modela i kreiranje kvalitativnog modela odlučivanja. Neke promenljive koje smo uočili u prethodnom koraku možda ne utiču na odluku o lečenju i mogu se izostaviti. Nekada se može desiti da se dve promenljive spoje u jednu ili da se razdvoje.
3. Određivanje verovatnoća. Podaci o verovatnoći se mogu dobiti na osnovu analize literature, baza podataka o pacijentima ili subjektivne procene eksperta.
4. Određivanje korisnosti (utility). Napravi se skala od 0 (tragičan ishod) do 1 (potpuno ozdravljenje) i ostale moguće ishode odredimo u toj skali. Ponekad skalu može da odredi sam ekspert, a postoje i slučajevi gde se moraju konsultovati pacijenti ili roditelji pacijenta jer ljudi mogu imati različite preferencije.
5. Verifikacija i usavršavanje modela. Verifikacija se radi tako što se stručnjacima iz oblasti (u ovom slučaju lekarima) daju neki slučajevi i da oni izvrše dijagnozu i predlože lečenje. Zatim se njihove odluke porede sa odlukama sistema. Ako se odluke ne poklapaju najbolje onda se vrši analiza koji deo sistema uzrokuje problem i popravljaju se.
6. Senzitivna analiza. U ovom važnom koraku proverava se koliko su odluke osetljive na neke male promene u vrednostima promenljivih i korisnosti. Ako male promene dovode do velikih razlika u odlučivanju, treba nešto ispravljati. Ako male promene ne menjaju u velikoj meri odlučivanje, znači da je sistem u velikoj meri pouzdan. Ovaj korak je značajan zbog toga što se glavna kritika korišćenja verovatnoće u ekspertnih sistemima odnosi na činjenicu da je teško odrediti numeričku vrednost za verovatnoću. Senzitivna analiza otkriva da mnogi brojevi moraju da se odrede samo otpilike.

Donošenje kompleksnih odluka

Do sada smo posmatrali problem donošenja jedne odluke u okruženju u kojem je funkcija korisnosti bila dobro poznata na osnovu ishoda jedne akcije. Ovde ćemo analizirati problem sekvencijalnog odlučivanja u kom korisnost zavisi od sekvence odluka.

Prtpostavimo da se agent nalazi u okruženju koje je prikazano na slici 8.4. Početna pozicija je označena sa START i svakom koraku agent bira jednu od mogućih akcija, a to je da ode Gore, Dole, Levo, Desno. Agentova interakcija sa okruženjem se završava kada on dođe do nekog od završnih stanja +1 ili -1. Pretpostavićemo da se okruženje može potpuno sagledati.



Slika 8.4

Ako bi okruženje na slici 8.4 bilo determinističko, rešenje bi bilo jednostavno: [Gore,Gore,Desno,Desno,Desno]. Međutim, ovo okruženje nije determinističko i ishodi akcija agenta nisu pouzdani. Agent se kreće stohastički i to tako da ga izabrana akcija sa verovatnoćom 0.8 šansi vodi u odgovarajuće polje, a sa 0.1 šansi ga pomera levo i sa 0.1 šansi ga pomera desno (slika 8.4 (b)). Ako agent udari u zid on ostaje u istom polju. U ovakvom okruženju sekvenca akcija [Gore,Gore,Desno,Desno,Desno] dostiže ciljno polje (4,3) sa verovatnoćom $0.8^5=0.32768$ plus verovatnoća da se birajući iste akcije sa druge strane dođe do cilja, a to je $0.1^4 \cdot 0.8$.

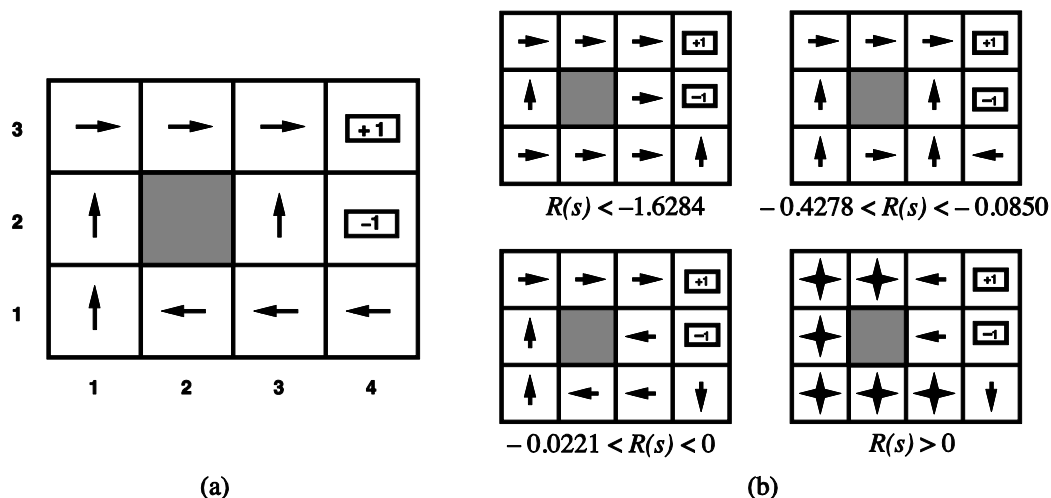
Tranzicioni model kod ovakvih okruženja posmatračemo stohastički, $P(s'|s,a)$ je verovatnoća da se iz stanja s dostigne s' ako se izvrši akcija a . Pretpostavićemo da su promene okruženja takve da verovatnoća da se iz s dođe do s' zavisi samo od s , a ne od istorije stanja.

Koja je funkcija korisnosti agenta? Kako je problem odlučivanja sekvencijalni, funkcija korisnosti će zavisiti od sekvence stanja - istorije okruženja. Moguće je napraviti uopštenu funkciju korisnosti, ali ćemo za sada posmatrati pojednostavljen slučaj i u svakom stanju s agentu dati neku nagradu (reward) $R(s)$ koja može biti pozitivna ili negativna. Za primer okruženja na slici 8.4 nagrada je -0.04 u svim stanjima osim završnim, koja imaju nagrade +1 i -1. Korisnost istorije okruženja je suma svih dobijenih nagrada. Na primer, ako agent dostigne stanje +1 posle 10 koraka, njegova ukupna korisnost je 0.6. Negativna nagrada navodi agenta da cilj dostigne brzo. Opisano okruženje predstavlja stohastičku generalizaciju rešavanja problema pretraživanjem.

Sledeće pitanje je kako izgleda rešenje ovakvog problema. Bilo koji fiksni niz akcija neće rešiti problem jer izvršavajući taj niz agent ne mora sigurno da dođe do završnog stanja. Znači da bi rešenje trebalo da specificira šta agent treba da uradi u svakom stanju u kom se može naći. Rešenja ovog tipa nazivaju se polise. Polisa se najčešće označava sa π , a sa $\pi(s)$ označavaju se akcije koje se preporučuju iz stanja s prema polisi π . Ako agent ima kompletnu polisu on će znati u bilo kom stanju šta da radi bez obzira na ishod akcije.

Kvalitet polise se meri preko očekivanih korisnosti svih mogućih istorija okruženja koji su definisani polisom. Optimalna polisa je polisa koja ima najveću očekivanu korisnost i označava se sa π^* . Agent odlučuje šta će da uradi tako što utvrdi u kome je stanju s i zatim izvršava akciju $\pi^*(s)$.

Optimalna polisa za okruženje sa slike 8.4 prikazana je na slici 8.5 (a). Pošto je cena koraka prilično mala u odnosu na kaznu za ulazak u (4,2), optimalna polisa za stanje (3,1) je neefikasna, jer zbog rizika da se uđe u (4,2) preporuka je da se ide okolo.



Slika 8.5

Balans rizika i nagrade zavisi od vrednosti $R(s)$ za stanja koja nisu završna. Slika 8.4 prikazuje različite optimalne polise u odnosu na četiri različita opsega $R(s)$. Kada je $R(s) < -1.6284$, agent u svakom stanju gleda kako da što pre izađe, pa makar i ušao u -1. kada je $-0.4278 \leq R(s) \leq -0.0850$ agent će birati najkraći put do +1, pa makar i rizikovao da slučajno ode u stanje sa -1. U slučaju $-0.0221 < R(s) < 0$ optimalna polisa ne preporučuje nikakav rizik. I konačno ako je $R(s) > 0$, agent izbegava izlaz iz okruženja.

Pažljivo balansiranje između rizika i nagrade je karakteristika rešavanja ovakvih problema.

Poglavlje 9

Osnove mašinskog učenja

Kažemo da agent uči ako on unapređuje svoj učinak u rešavanju budućih zadataka nakon opservacije sveta koji ga okružuje. Zašto je potrebno da agenti uče? Zašto nije dovoljno da dizajner jednostavno implementira sva potrebna unapređenja? Postoje tri razloga, jedan je što dizajner ne može da predvidi sve situacije u kojima se agent može naći (na primer robot u lavirintu). Drugi je što dizajner ne može da predvidi kako će se zapažanja menjati u toku vremena (program za predviđanje sutrašnjih cena akcija na berzi mora da nauči da se prilagodi kada se uslovi menjaju). I treći je da dizajner nekad ne zna kako da implementira rešenje nekog problema (većina ljudi dobro prepoznaje lice, ali ne bi znala da napravi program koji to radi).

Pojam mašinskog učenja odnosi se na izgradnju prilagodljivih računarskih sistema čija je jedna od osnovnih uloga poboljšanje performansi. Mašinsko učenje ima veliku praktičnu primenu, zbog čega je praksa u ovoj oblasti napredovala brže od teorije. Poznati projekti mašinskog učenja su:

- ALVINN - Autonomous Land Vehicle In a Neural Network, sistem koji uči da upravlja vozilom tako što posmatra ljude kako voze; prima informacije preko video kamera sa vozila kojima upravlja čovek, naučio je da vozi javnim putem u prisustvu drugih vozila brzinom do 110km/h; uspešno je vozio na putu dužine 140km

- TD-Gammon - sistem za igranje Backgammon-a koji je igrajući protiv sebe i sa drugim igračima naučio da igra na nivou švedskih šampiona; koristi algoritam sličan MINIMAX-u a uči funkciju evaluacije.

Mašinsko učenje ima veliku primenu u oblasti prepoznavanja govora i prepoznavanja rukom pisanog teksta.

Na osnovu povratnih informacija sa kojima agent raspolaže razlikujemo dve vrste učenja:

- nenadgledano učenje (unsupervised learning)
- nadgledano učenje (supervised learning)

Nenadgledano učenje podrazumeva pronalaženje paterni u ulaznim podacima u okruženju gde ne postoji povratna informacija (agent nema informacije iz spoljašnjeg sveta u vidu nagrade ili kazne). Najčešći oblik ove vrste učenja je klasterovanje: detektovanje potencijalno korisnih klastera iz datih primera. Pojam klasterovanja se odnosi na grupisanje objekata u grupe (klustere) tako da su objekti koji pripadaju jednom klasteru sličniji jedni drugima (prema nekoj osobini) i razlikuju se od objekata koji pripadaju drugom klasteru. Jedan primer primene klasterovanja je redukcija skupa boja slike. Pikseli slike se mogu grupisati klasterovanjem po njihovoj blizini u RGB prostoru boja, a potom se iz svakog klastera može izabrati po jedna boja koja bi ga predstavljala i kojom bi bili obojeni svi pikseli koji pripadaju tom klasteru.

Nadgledano učenje vrši analizu datih primera parova ulaz-izlaz i izvođenje funkcije koja preslikava ulaz na izlaz. Na primer, ulaz mogu da budu digitalne fotografije, a izlaz informacija šta je na fotografiji, ulaz mogu biti ručni zapisi slova a izlaz "prepoznato slovo".

9.1 Nadgledano učenje

Nadgledano učenje podrazumeva neku "obuku" koja se sastoji od N parova ulaz-izlaz

$$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N),$$

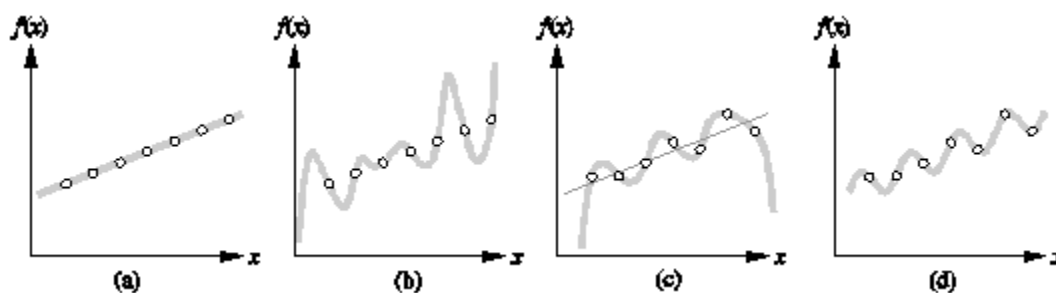
gde je svako y_i dobijeno nekom nepoznatom funkcijom $y=f(x)$.

Zadatak učenja je da otkrije funkciju h koja predstavlja aproksimaciju funkcije f .

Ovde x i y mogu biti bilo kakve vrednosti, ne moraju biti brojevi. Funkcija h se naziva hipoteza. Učenje predstavlja pretraživanje prostora mogućih hipoteza (\mathcal{H}) čiji je cilj da se pronađe hipoteza koja najviše odgovara i koja će važiti i za nove primere izvan obuke. Da bi se izmerila tačnost hipoteze, daje se skup testova koji se sastoje od primera koji nisu obuhvaćeni obukom. Kažemo da hipoteza vrši dobru generalizaciju ako pravilno predviđa vrednosti za y za nove primere. Za hipotezu kažemo da je konzistentna ako se ona slaže sa funkcijom f u svim posmatranim primerima.

Kada je izlaz y konačan skup vrednosti (na primer sunčano, oblačno, kišovito) problem učenja se naziva klasifikacija. Ako postoje samo dva moguća izlaza onda je to logička ili binarna klasifikacija. Kada je y neki broj (na primer temperatura u celzijusima) onda se problem učenja naziva regresija.

Na slici 9.1 prikazane su dva slučaja funkcije f sa po dve moguće hipoteze. Za prostor hipoteza ćemo posmatrati polinomne funkcije bilo kog stepena. Prvi slučaj funkcije f prikazan je na slikama a i b na kojima je h polinomna funkcija stepena 1 na slici a i stepena 7 na slici b. Obe ove hipoteze su konzistentne. Postavlja se pitanje kako izabrati između dve konzistentne hipoteze. Jedan princip je da se izabere "jednostavnija" hipoteza. U posmataranom slučaju (a i b) je očigledno da je prava linija jednostavnija od krivulje, ali postoje slučajevi gde nije jednostavno utvrditi koja hipoteza je jednostavnija.



Slika 9.1

Ako posmatarmmo drugi primer na slici 9.1 (c i d) videćemo da ovde ne postoji konzistentna hipoteza u obliku prave linije. Potreban je polinom šestog stepena (c). Međutim, kako postoji samo sedam tačaka, polinom dobijen na osnovu sedam poznatih tačaka neće vršiti dobru generalizaciju. Na slici (c) je prikazana i prava linija koja predstavlja hipotezu koja nije konzistentna ali vrši bolju generalizaciju.

Uopšteno govoreći, mora se praviti kompromis između kompleksnih hipoteza koje odgovaraju podacima iz "obuke" i jednostavnih hipoteza koje vrše bolju generalizaciju. Na slici 9.1 (d) prikazana je hipoteza koja uključuje i sinusnu funkciju (prostor hipoteza je proširen sa uključivanjem sinusa u polinom). Ovakva funkcija najbolje generalizuje dati skup podataka, što pokazuje kako je značajan izbor prostora hipoteza.

9.2 Klasifikacija

Klasifikacija predstavlja razvrstavanje nepoznatih instanci u kategorije. Na primer, klasifikacija bankovskih transakcija na rizične i nerizične, klasifikacija elektronske pošte na onu koja sadrži spam i koja ne sadrži spam, prepoznavanje autorstva tekstova na osnovu rukopisa. U navedenim primerima, svaka instanca se može opisati preko nekog skupa atributa. Problem klasifikacije je određivanje kojoj klasi pripada instanca na osnovu vrednosti atributa. Formalno, klasifikacija je aproksimacija ciljne funkcije koja svakoj instanci dodeljuje oznaku klase kojoj ta instanca pripada.

Postoji veliki broj metoda za rešavanje ovog problema, a neke od njih su:

- metoda zasnovana na instancama (eng. instance based classification)
- učenje stabla odlučivanja (eng. decision tree induction)
- metoda potpornih vektora (eng. support vector machines)
- metode bayesove klasifikacije (eng. bayesian classification)

U nastavku će biti opisane prve dve metode.

9.2.1 Metoda zasnovana na instancama

Osnovna karakteristika metode zasnovane na instancama je da ne grade eksplicitni model podataka u vidu neke funkcije kao što to radi većina metoda mašinskog učenja. Samim tim, klasifikacija se ne vrši na osnovu modela ili funkcije već na osnovu izabranog skupa primera za obuku. Postoje dva poznata koncepta koja se koriste u metodama zasnovanim na instancama, to su metoda n-najbližih suseda (n-nearest neighbours) i n-grami.

Metoda n-najbližih suseda zasniva se na principu da se nepoznata instanca klasifikuje u klasu čije su instance najbližije nepoznatoj. Koncept sličnosti se formalizuje preko funkcije rastojanja. Intuitivno je jasno da manje rastojanje predstavlja veću sličnost. Kako postoje različite funkcije rastojanja potrebno je izabrati onu koja je relevantna za posmatrani domen. Za slučaj $n=1$ klasifikovanje instance u klasu iz koje potiče instanca trening skupa koja joj je najbliža. U opštem slučaju pronalazi se n instanci u trening skupu i nepoznata instanca se dodeljuje klasi čiji se elementi najčešće javljaju među pronađenih n suseda.

N-grami. Metode mašinskog učenja su najčešće formulisane tako da se jednostavno primenjuju na numeričke podatke, ali teško na podatke u nekom drugom obliku. Primeri ovakvih problema su klasifikacija tekstova ili proteinskih sekvenci, a oni se rešavaju korišćenjem koncepta n-grama.

Ako je data niska $S=s_1s_2\dots s_N$ nad nekom azbukom, i N je pozitivan ceo broj, n -gram niske S (za $n \leq N$) je bilo koja podniska susednih simbola dužine n . N -gramski profil niske je lista uređenih parova (n -gram, frekvencija) gde je frekvencija izračunata u odnosu na sve n -grame niske. Ovi profili predstavljaju reprezentaciju pogodnu za metode klasifikacije i koriste se kombinovano sa metodom najbližih suseda. Uspešno su korišćeni u različitim primenama koje uključuju prepoznavanje autorstva tekstava, prepoznavanje jezika na kom je tekst napisan.

9.2.1 Učenje stabla odlučivanja

Učenje stabla odlučivanja su jedan od najjednostavnijih, ali i jedan od najuspešnijih formi mašinskog učenja. Prvo ćemo opisati reprezentaciju (prostor hipoteza), a zatim ćemo videti koje su tehnike za učenje najbolje hipoteze.

Reprezentacija stabla odlučivanja

Stablo odlučivanja je funkcija koja kao ulaz uzima vektor vrednosti atributa, a kao rezultat vraća neku odluku (jednu vrednost). Ulazne i izlazne vrednosti mogu biti diskretne ili kontinualne. Ovde ćemo posmatrati probleme gde su ulazne informacije diskretne vrednosti a izlaz ima dve moguće vrednosti, odnosno radi se o logičkoj klasifikaciji u kojoj se ulaz klasifikuje kao tačan (pozitivan primer) ili netačan (negativan primer).

Stablo odlučivanja donosi odluku na osnovu serije testova. Svaki čvor koji nije list predstavlja jedan test na jedan atribut A_i iz ulaznog skupa atributa. Grane koje izlaze iz čvora predstavljaju moguće vrednosti atributa, $A_i=v_{ik}$. Svaki list predstavlja izlaznu vrednost funkcije.

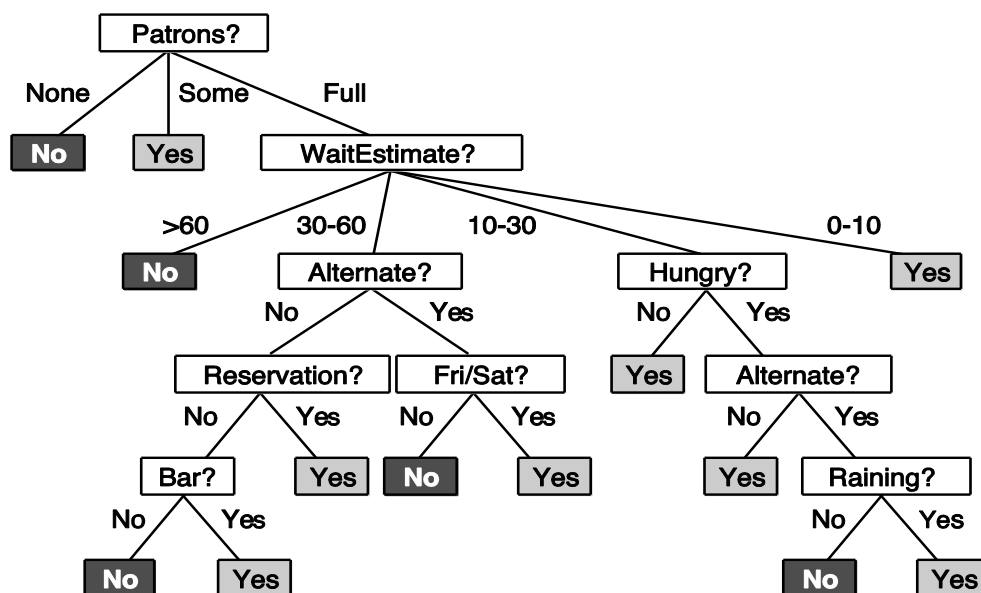
Kao primer ćemo napraviti stablo odlučivanja koje odlučuje da li ćemo da čekamo na sto u restoranu. Cilj je da se nauči definicija ciljnog predikata *TrebaČekati* (*WillWait*). Atributi koje ćemo posmatrati kao ulaz su:

1. *Alternative*: da li u blizini postoji pogodan restoran
2. *Bar*: da li restoran ima pogodan bar u kom se može sačekati na sto
3. *Fri/Sat*: tačno ako je petak ili subota
4. *Hungry*: koliko smo gladni.
5. *Patrons*: koliko ima ljudi u restoranu (vrednosti su *Nine*, *Some* and *Full*)
6. *Price*: opseg cena restorana (\$, \$\$ i \$\$\$)
7. *Raining*: da li napolju pada kiša.
8. *Reservations*: da li imamo rezervaciju
9. *Type*: vrsta restorana (French, Italian, Thai, burger)

10. *WaitEstimate*: procena koliko treba čekati (0-10 minuta, 10-30, 30-60, >60)

Svaki od navedenih atributa ima mali skup mogućih vrednosti i svi uzimaju vrednosti iz diskretnog skupa.

Na slici 9.2 prikazano je jedno moguće stablo odlučivanja za ovaj problem. U ovom stablu izostavljena su dva atributa *Type* i *Price*.



Slika 9.2

Logičko stablo odlučivanja je logički ekvivalentno sa izkazom da je ciljni atribut tačan ako i samo ako skup ulaznih atributa zadovoljava jednu od putanja koja vodi do lista za vrednošću true (yes), što se u logici može zapisati kao:

$$Goal \Leftrightarrow (Path1 \vee Path2 \vee \dots).$$

Gde je *Path* konjukcija testova atributa-vrednost koji se nalaze na posmatranoj putanji. Na primer putanja koja se nalazi skroz desno se može zapisati sa $(Patrons = Full \wedge WaitEstimate = 0 - 10)$. Ovakva forma je u disjunktivnoj normalnoj formi, što znači da se svaka funkcija iskazne logike može izraziti preko stabla odlučivanja.

Velika grupa problema se može presizno i elegantno predstaviti stablom odlučivanja. Međutim, postoje i funkcije koje za koje je to nemoguće. Na primer, funkcije koja vraćaju tačno ako i samo ako polovina ulaznih atributa ima vrednost tačno zahteva stablo odlučivanja eksponencijalne veličine. Drugim rečima, stabla odlučivanja su pogodna za određene grupe problema, ali ne za sve. Postavlja se pitanje da li postoji reprezentacija koja je pogodna za rešavanje svih problema i odgovor je ne.

Kreiranje stabla odlučivanja iz primera

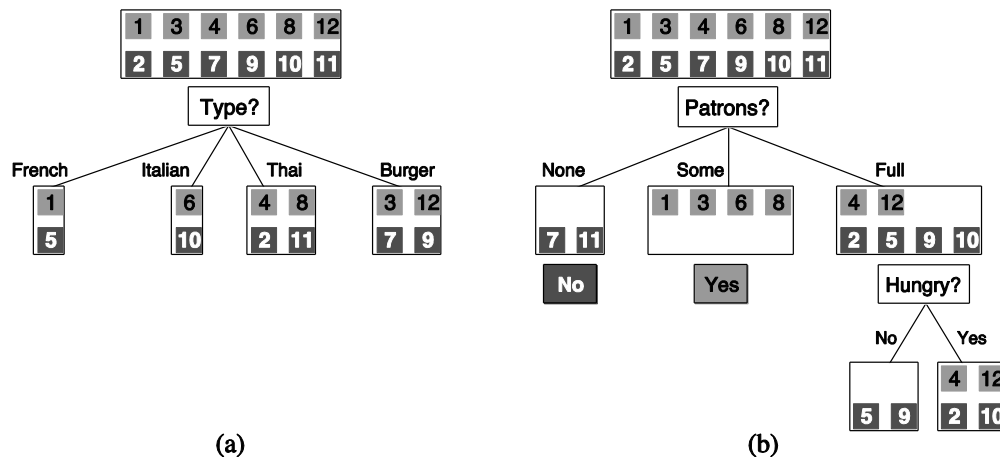
Primer logičkog stabla odlučivanja sastoji se od parova (\mathbf{x}, y) gde je \mathbf{x} vektor vrednosti ulaznih atributa, a y je Boolean vrednost koja predstavlja izlaz. Obuka koja sadrži 12 primera data je na slici 9.3. Pozitivni primeri su oni za koje je vrednost u poslednjoj koloni T (true) ($\mathbf{x}_1, \mathbf{x}_3, \dots$), a negativni oni u kojima je ta vrednost F (false) ($\mathbf{x}_2, \mathbf{x}_5, \dots$).

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>WillWait</i>
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0–10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30–60	T

Slika 9.3

Zadatak je da se napravi najmanje moguće stablo koje je konzistentno sa datim primerima. Algoritam koji se koristi za rešavanje ovog problema je DECISION-TREE-LEARNING algoritam koji koristi gramzivu strategiju "podeli pa vladaj" koja predviđa da se uvek prvo testira najznačajniji atribut (atribut koji se testira naziva se testni atribut). Ovaj test deli problem na manje potprobleme koji se zatim rešavaju rekursivno. Pod najznačajnijim atributom podrazumeva se atribut koji najviše utiče na klasifikaciju, odnosno izlaz.

Na slici 9.4 prikazana je analiza važnosti dva atributa, to su atribut *Type* i atribut *Patron*. Na slici 9.4 a možemo videti da *Type* atribut nema veliku važnost jer sve četiri mogućnosti imaju jednak broj negativnih i pozitivnih izlaza. Sa druge strane atribut *Patrons* je prilično važan atribut, jer ako je njegova vrednost *None* ili *Some* imamo jasan rezultat. Ako je vrednost *Full* onda imamo mešoviti rezultat. Uopšteno govoreći, prvi korak je da se izabere jedan atribut i za svaku njegovu moguću vrednost dobijamo podskup primera. Svaki podskup se posmatra kao novi problem učenja, odnosno kreiranje stabla odlučivanja na osnovu datih primera.

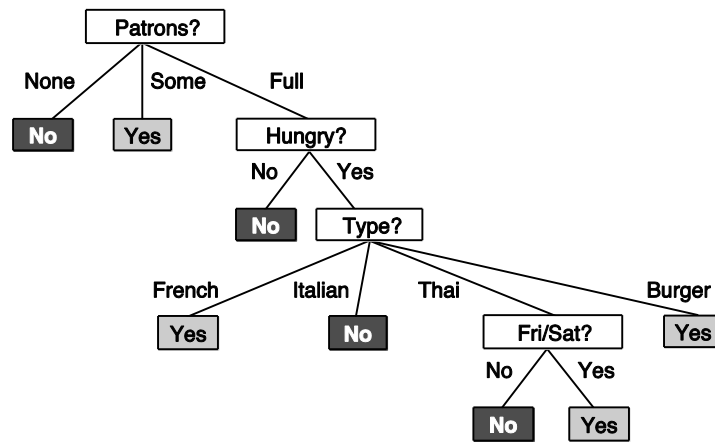


Slika 9.4

Postoje četiri slučaja koje treba razmotriti u rekurziji:

1. Ako su preostali primeri svi pozitivni (ili negativni) onda smo završili, možemo da odgovorimo sa Da ili Ne (slika 9.4 b)
2. Ako ima pozitivnih i negativnih primera u nekom podskupu, tada se bira najbolji atribut za njihovo razdvajanje. na slici 9.4 (b) to je atribut *Hungry*.
3. Ako nema preostalih primera, to znači da ne postoji primer koji zadovoljava posmatrani skup vrednosti atributa. Rezultat dobijamo tako što posmatramo sve primere kojima je konstruisan roditeljski čvor i uzimamo vrednost koju ima većina primera.
4. Ako nema preostalih atributa, ali ima preostalih pozitivnih i negativnih primera, to znači da ti preostali primeri imaju isti opis, ali različitu klasifikaciju. Ovo se može dogoditi u slučaju da postoji greška u podacima, ako je domen nedeterministički ili ako ne možemo da dobijemo informaciju koja bi nam omogućila da razlikujemo dva posmatrana primera.

Treba primetiti da je skup primera sa kojim raspolazemo ima ključnu ulogu za konstruisanje stabla. Opisani algoritam će nam za skup primera koji je prikazan na slici 9.3 dati stablo prikazano na slici 9.5. Ovo stablo se svakako razlikuje od originalnog stabla sa slike 9.2, ali iz ovoga ne treba zaključiti da algoritam nije dobro odradio posao. Suština je u tome što algoritam posmatra samo raspoložive primere, a ne celu funkciju, i daje hipotezu koja je konzistentna sa svim primerima. Algoritam nije uzeo u obzir attribute *Rain* i *Reservation* jer je uspeo da klasifikuje sve primere i bez njih. Sa druge strane, algoritam je detektovao zavisnost odluke od vrste restorana, što u originalnom stablu nije podržano. Opisani algoritam učenja može da napravi određene greške u slučajevima za koje nema primere. Na primer, u skupu primera (slika 9.3) ne postoji slučaj u kom je predviđeno vreme čekanja 0-10, a restoran je pun, zbog čega je algoritam izabrao false za slučaj kad je *Hungry* false iako bi većina ljudi izabrala da čeka. Sa više primera algoritam bi izbegao ovu grešku.



Slika 9.5

Poglavlje 10

Obrada prirodnog jezika

Postoje dva osnovna razloga zašto želimo da osposobimo agenta da razume prirodni jezik, a to su komunikacija sa čovekom i drugi je usvajanje znanja (eng. knowledge acquisition) iz izvora napisanim na prirodnom jeziku (na primer Web). Ako usvajanje znanja posmatramo kao potragu za informacijama, razlikujemo sledeće oblasti: klasifikacija teksta, pribavljanje informacija (information retrieval) i ekstrakcija informacija.

Prirodnim jezikom smatramo jezik kojim govore i pišu ljudi u svakodnevnoj komunikaciji (srpski, engleski, francuski). Jezik možemo posmatrati kao sistem koji sadrži skup simbola i skup pravila za njihovo kombinovanje (gramatika).

Značajan faktor u oblasti prepoznavanja prirodnog jezika je korišćenje modela jezika tj. modela koji predviđaju verovatnoću pojavljivanja jezičkih izraza.

10.1 Modeli jezika

Jezik se može definisati kao skup stringova. Formalni jezici (kao što su Java ili Python) imaju precino definisan model i za svaki izraz možemo reći da li pripada tom jeziku ili ne (na primer. $\text{int } i=0$). Formalni jezici se specificiraju skupom pravila koji se nazivaju gramatika. Formalni jezici takođe propisuju pravila kojima se izrazima dodeljuje smisao, odnosno semantika. Tako je smisao izraza "2+2" jednaka 4, odnosno znak + se interpretira kao operacija sabiranja.

Prirodni jezici, kao što su Engleski ili Srpski se ne mogu u potpunosti opisati gramatičkim pravilima, kao što je to slučaj sa formalnim jezicima. Umesto toga, modeli prirodnih jezika se mogu definisati kao distribucija verovatnoća nad rečenicama ili izrazima. To znači da, umesto da pitamo da li reč "pas" pripada nekom jeziku, pitamo $P(S=\text{pas})$, odnosno koja je verovatnoća da proizvoljna rečenica u prirodnom jeziku sadrži samo reč pas.

Problem sa obradom prirodnog jezika je u njegovoj dvosmislenosti i opet se ne može govoriti o jedinstvenom značenju rečenice, već o verovatnoći mogućih značenja. Još jedan problem obrade prirodnog jezika je njegova veličina. Prirodni jezici su veoma veliki i stalno se menjaju. U nastavku će biti opisani neki poznati modeli jezika.

N-gram modeli nad karakterima

Pisani tekst na prirodnom jeziku sastoji se od karaktera – slova, cifara, znakova interpunkcije, razmaka, i sl. Jedan od najjednostavnijih modela jezika jeste distribucija verovatnoća pojave određene sekvence karaktera u jeziku. Oznakom $P(c_{1:N})$ označavaćemo verovatnoću pojave sekvence od N karaktera od c_1 do c_N . Tako na primer, ako posmatramo kolekciju Web stranica, važi da je $P(\text{"the"})=0.027$ a $P(\text{"zgq"})=0.000000002$. Sekvenca simbola dužine n, naziva se n-gram a specijalni slučajevi su unigram za 1-gram, bigram za 2-gram i trigram za 3-gram. Model koji definiše verovatnoće pojave n-grama naziva se n-gram model. N-gram modeli se mogu definisati nad karakterima, sekvencama reči ili slogovima.

Za trigram model jezika u jeziku koji ima 100 karaktera postoji oko milion mogućih sekvenci od tri karaktera odnosno treba izračunati verovatnoće pojavljivanja za oko milion elemenata. Ove verovatnoće se mogu precizno izračunati (proceniti) brojanjem sekvenci karaktera u količini teksta od 10 miliona karaktera i više (tekst za obuku). Celokupna količina teksta koja se koristi za obuku naziva se korpus.

N-gram model nad karakterima koristi se u oblasti prepoznavanja jezika (za dati tekst prepoznaje na kom jeziku je napisan). Kompiuterski sistemi danas vrše prepoznavanje jezika sa uspešnošću od 99%. Greške nastaju kod sličnih jezika kao što su švedski i norveški.

Ostale oblasti u kojima se koristi ovaj model je ispravljanje grešaka nastalih kucanjem (spelling correction), klasifikacija teksta na osnovu žanra, i prepoznavanje imenovanih entiteta u tekstu i određivanje kojoj klasi pripadaju (na osnovu sufiksa na primer).

Mogući problem sa n-gram modelima je da analiza korpusa daje samo procenu verovatnoća pojave n-grama. Na primer, za trigram „ th“ bilo koji korpus teksta na engleskom jeziku će dati veliku verovatnoću, dok će za trigram „ ht“ verovatnoća pojavljivanja za većinu korpusa biti 0. Da li to znači da će i reč „ http“ imati verovatnoću pojavljivanja 0? Ovo je problem generalizacije, jer hoćemo da na osnovu nekog korpusa izvršimo generalizaciju nad svim ostalim tekstovima. Ako u korpusu nad kojim se trenira nije postojala reč „ http“ ne treba zaključiti da ta reč ne postoji u jeziku. Ovaj problem se rešava tako što se u modelu jezika sekvencama karaktera koje imaju verovatnoću pojavljivanja 0 dodeli neka mala verovatnoća veća od nule, a ostale verovatnoće se podešavaju tako da zbir ostane 1. Ovaj proces se naziva glačanje (smoothing) modela jezika.

N-gram modeli nad rečima

Pored n-grama nad karakterima, u praksi su prisutni i n-gram modeli nad rečima, gde se umesto karaktera posmatraju reči nekog jezika. Osnovna razlika između ova dva modela je što je u modelu nad rečima rečnik koji se posmatra mnogo veći (u jednom jeziku ima mnogo više reči u odnosu na broj karaktera, obično je taj odnos 100 prema nekoliko desetina hiljada ili čak milion). U nekim jezicima nije tako očigledno šta predstavlja reč, kao što je kineski, na primer.

Još jedna bitna razlika n-gram modela nad rečima u odnosu na n-gram model nad karakterima je što moramo da uzimamo u obzir stalno povećavanje rečnika, odnosno uvođenje novih reči. Ovaj problem se često rešava tako što se u rečnik uvede jedna nova reč, na primer <UNK> (unknown word). Za ovu reč se može izračunati n-gram na sledeći način: prolazimo kroz korpus za obuku i prvo pojavljivanje neke nepoznate reči zamenimo simbolom <UNK>. Sve ostale pojave nepoznate reči ostaju nepromenjene. Zatim se izračuna broj n-grama na standardan način a simbol <UNK> se posmatra kao i svaka druga reč. Zatim, kada se nepoznata reč pojavi u testnom skupu (odnosno tekstu van korpusa) za njenu verovatnoću uzimamo verovatnoću simbola <UNK>. Nekada se različiti simboli koriste za različite klase reči, na primer <NUM> za brojeve i <EMAIL> za email adrese.

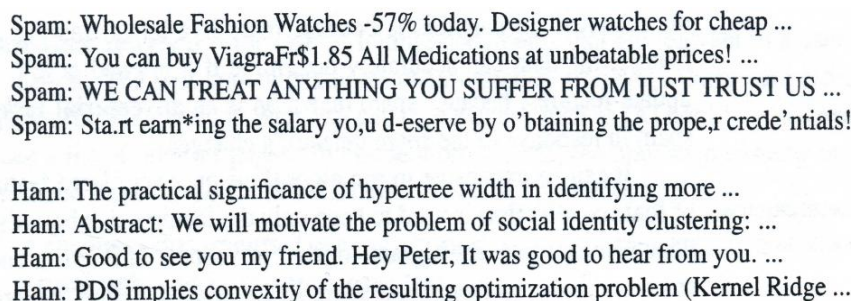
N-gram modeli teksta nad karakterima ili rečima predstavljaju osnovu za ostale zadatke obrade prirodnog jezika, kao što su klasifikacija teksta, pribavljanje informacija i ekstrakcija informacija.

10.2 Klasifikacija teksta

Klasifikacija teksta, često se naziva i kategorizacija teksta, predstavlja proces određivanja kojoj klasi (iz nekog predefinisano skup klasi) pripada neki dati tekst. Moguće klasifikacije su identifikacija jezika, određivanje žanra, klasifikacija neke recenzije za film ili knjigu kao pozitivnu ili negativnu (sentimentalna analiza), detekcija spamova i sl.

Detekcija spamova je proces klasifikacije e-mail poruka kao spam ili ne-spam (ham). Ova detekcija spamova je problem nadgledanog učenja a kao trening skup se uzimaju emailovi iz inboxa kao primer za ham, a emailovi iz spam foldera kao primer za spam.

Na slici 10.1 prikazani su primeri nekoliko spam i ham poruka. Iz tih primera možemo zaključiti šta bi bili dobri indikatori za klasifikaciju email poruka. Na primer u n-gram modelu nad rečima pojavljivanje izraza „for cheap“ ili „you can buy“ može da ukaže na spam, mada verovatnoća njihovog pojavljivanja u ham porukama nije nula. Na nivou karaktera možemo da upotrebimo rezon da su spam poruke često ispisane svim velikim slovima i da su znaci interpukcije često pomešani sa tekstom.



Spam: Wholesale Fashion Watches -57% today. Designer watches for cheap ...
Spam: You can buy Viagra for \$1.85 All Medications at unbeatable prices! ...
Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
Spam: Start earning the salary you deserve by obtaining the proper credentials!

Ham: The practical significance of hypertree width in identifying more ...
Ham: Abstract: We will motivate the problem of social identity clustering: ...
Ham: Good to see you my friend. Hey Peter, It was good to hear from you. ...
Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge ...

Slika 10.1 Primeri spam i ham poruka

Postoje dva komplementarna načina rešavanja problema detekcije spamova, jedan je zasnovan na modelu jezika, a jedan na nadgledanom mašinskom učenju.

U pristupu koji je zasnovan na modelu jezika potrebno je odrediti (na osnovu korpusa za obuku) jedan n-gram model za $P(\text{Poruka}|\text{spam})$, odnosno verovatnoću pojavljivanja n-grama u spam porukama i jedan n-gram model za $P(\text{Poruka}|\text{ham})$ odnosno verovatnoću pojavljivanja n-grama u ham porukama. Zatim se verovatnoća da je neka poruka spam ili ham računa korišćenjem Bajesovog pravila na sledeći način:

$$\operatorname{argmax}_{c \in \{\text{spam}, \text{ham}\}} P(c|\text{poruka}) = \operatorname{argmax}_{c \in \{\text{spam}, \text{ham}\}} P(\text{poruka}|c)P(c)$$

Gde se $P(c)$ procenjuje prebrojavanjem spam i ham poruka iz korpusa obuke.

U pristupu koji se zasniva na mašinskom učenju poruka se predstavlja kao skup parova osobina/vrednosti i preminjuje se neki od algoritama klasifikacije. Ova dva pristupa (model jezika i mašinsko učenje) se mogu učiniti komplementarnim tako što se n-grami posmatraju kao osobine. U slučaju unigrama na primer osobine su reči iz jezika, a vrednosti broj pojavljivanja reči u poruci. Na ovaj način dobijamo veliki broj osobina (oko 100 000), ali za većinu email poruka vrednost većine osobina će biti nula. U slučaju

unigramama posmatra se samo pojava reči, ali ne i redosled u kom se reči pojavljuju, što može da se prevaziđe sa n-gramima većeg reda, ali se kod njih znatno povećava broj osobina.

Pored n-grama u identifikaciji spamova mogu se dodati još neke osobine, kao što su pojava URL adresa ili slika u poruci, broj spamova ili hamova koji su prethodno stigli od istog pošiljaoca, vreme kada je poslata poruka, i sl. Izbor osobina u detekciji spamova je najbitniji faktor uspešnosti, mnogo je bitniji od odabira algoritma za obradu tih osobina. Pored toga, potrebno je stalno ažurirati skup osobina, jer problem detekcije spamova ima i takmičarskih elemenata (eng. adversarial) jer se proizvođači spamova stalno prilagođavaju programima za detekciju i pokušavaju da ga „pobede“.

10.3 Pribavljanje informacija (Information retrieval)

Pribavljanje informacija (Information retrieval-IR) je zadatak pronalaženja dokumenata koji sadrže informacije potrebne korisniku. Najpoznatiji IR sistemi su Web pretraživači (na primer Google). IR sistemi se mogu okarakterisati sledećim osobinama:

1. Korpus dokumenata. Svaki sistem treba da odredi šta će posmatrati kao dokument: paragraf, stranicu, tekst na više stranica.
2. Upiti i upitni jezik. Upitom korisnik definiše koje informacije su mu potrebne. Upitni jezik može biti samo lista reči [AI books], fraza [“AI books”] ili može sadržati logičke operatore [AI AND book], nelogičke operatore [AI NEAR book] ili [AI book site:www.aaai.org].
3. Skup pogodaka. To je podskup dokumenata koje IR sistem „smatra“ relevantnim za upit.
4. Prezentacija skupa pogodaka. Ovo može biti jednostavno kao rangirana lista dokumenata ili neki komplikovani grafički prikaz rezultata.

Rani IR sistemi bili su zasnovani na logičkom modelu ključnih reči. Svaka reč u upitu je za neki dokument dobijala vrednost true ako se nalazi u tom dokumentu, odnosno false ako se ne nalazi. U ovom slučaju se upitni jezik mogao definisati kao jezik logičkih izraza nad osobinama dokumenata. Dokument se smatrao relevantnim za upit samo ako logički izraz daje vrednost true. Ovakav model je bio dovoljno jednostavan za implementaciju, međutim imao je i dosta mana od kojih je prva nepostojanje nikakvog rangiranja dokumenata po važnosti, zatim upitni jezik nije bio dovoljno intuitivan za običnog korisnika

IR scoring funkcija

Većina IR sistema napustila je logički model i opredelila se za model zasnovan na statistici broja reči. Jedan od poznatih ovakvih modela je BM25 scoring funkcija koja je nastala u projektu Okapi i koristi je veliki broj biblioteka za pretraživanje, između ostalog i open source projekat Lucene.

Scoring funkcija uzima dokument i upit i vraća numeričku vrednost (skor) koja je veća za relevantnija dokumenta. U BM25 funkciji skor je linearna kombinacija skorova koji ima svaka reč u upitu. Tri faktora utiču na određivanje skora:

- Frekvencija sa kojom se reči u upitu pojavljuju u dokumentu, na primer za upit [poljoprivreda u Vojvodini] dokumenta koja imaju veći broj pojavljivanja reči poljoprivreda imaće veću scoring funkciju.

- Inverzna frekvencija reči u dokumentima. Na primer reč „u“ se pojavljuje u velikom broju dokumenata, znači da ta reč ima veliku frekvenciju ali malu inverznu frekvenciju, što znači da ta reč nije tako važna kao reči „poljoprivreda“ i „vojvodina“
- Veličina dokumenta. Dokument od milion reči će imati veću frekvenciju za sve reči iz upita, a ne mora biti relevantan za taj upit. Manji dokumenti sa većom frekvencijom traženih reči su mnogo bolji kandidati.

Poboljšanja IR sistema

Postoji veliki broj načina za poboljšanje rada IR sistema i Web pretraživači stalno unapređuju svoje algoritme u skladu sa novim pristupima i u skladu sa povećanjem količine dostupnih informacija na Web-u.

Jedan od čestih poboljšanja je model koji bolje definiše uticaj veličine dokumenta na relevantnost, jer sistem koji koristi BM25 favorizuje dokumenta manje veličine.

Takođe, BM25 funkcija posmatra svaku reč nezavisno, a postoje reči koje su u nekoj korelaciji, na primer reč u množini, u drugom padežu ili sinonimi, takođe reč napisana velikim i malim slovima. Većina IR sistema izjednačava mala i velika slova, a neki koriste odsecanje sufiksa (eng. stemming) da bi se na primer u engleskom jeziku izjednačila množina i jednina. Dosta IR sistema koristi sinonime. I odsecanje sufiksa i sinonimi su korisni u pretraživanju, ali mogu i negativno da utiču na preciznost (na primer ako neko postavi upit [Tim Coach] ne želi da dobije u pogocima dokumente koji sadrže reč sofa što je sinonim za coach).

Konačno, veliko poboljšanje IR sistema moguće je korišćenjem metapodataka – podataka koji nisu deo samog teksta dokumenta, na primer ključne reči koje nisu vidljive u dokumentu. Na Web-u ključan izvor informacija predstavljaju linkovi između dokumenata.

10.4 Ekstrakcija informacija

Pojam ekstrakcija informacija (information extraction) odnosi se na proces usvajanja znanja iz nestruktuiranog teksta na prirodnom jeziku uočavanjem određenih klasa, objekata i veza između objekata. Jedan primer ovakvog zadatka je ekstrakcija struktuiranih adresa iz Veb stranica gde bi se dobila posebna polja za ulicu, broj, grad, poštanski broj.

Jedna vrsta ekstrakcije informacija odnosi se na ekstrakciju atributa nekog objekta. Ovaj pristup pretpostavlja da se ceo tekst odnosi na jedan objekat i zadatak je da se prepoznaju određeni atributi, odnosno da se iz nestruktuiranog teksta sa neke Veb stranice na primer "IBM ThinkBook970. Our price: \$399.00" dobiju strukturirani podaci {Proizvođač= IBM, Model=ThinkBook970, Cena=\$399.00}.

Ovakvi problemi se mogu rešavati definisanjem templejta (paterna) za svaki atribut koji treba prepoznati. Templejt se definiše regularnim izrazom (regex). Primer regularnog izraza za ekstrakciju cene: [\$] [0-9]+([.][0-9][0-9])?. Sintaksa regularnih izraza zavisi od okruženja (paket za rad sa regularnim izrazima u Javi je java.util.regex).

Jedan od čestih pristupa u ekstrakciji informacija je Skriveni Model Markova (Hidden Markov Model) koji se zasniva na verovatnoćama. Skriveni Markov Model pretpostavlja da sistem koji se obrađuje ima osobine Markovljevog procesa (lanca) sa skrivenim stanjima. U skrivenim stanjima imamo nekakve observacije (zapažanja) na osnovu kojih određujemo verovatnoću postojanja određenog skrivenog stanja. U domenu ekstrakcije informacija obzervacije su konkretne reči napisane na prirodnom jeziku, a skrivena stanja su koncepti koje treba prepoznati (na primer da li ta reč pripada delu rečenice koji je pre atributa koji prepoznavamo, da li je to baš taj atribut ili se nalazi posle tog atributa).

Postoje projekti koji se bave kreiranjem velikih baza znanja i ontologija ekstrakcijom informacija iz velikog korpusa (na primer 10 miliona Veb stranica). Ovakvi projekti mogu obuhvatiti neki specifični domen, ili mogu biti i generalni.

Takođe, primenom tehnika mašinskog učenja moguće je da sistem sam formira template za ekstrakciju informacija.

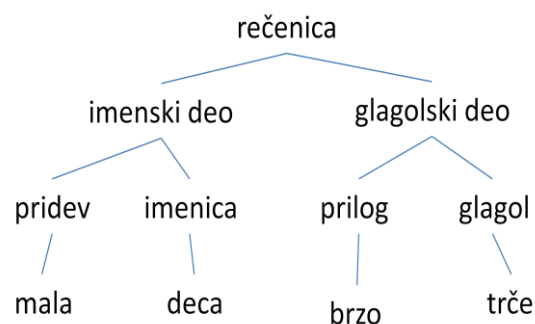
10.5 Prirodni jezik za komunikaciju

Jedan od razloga zašto se veštačka inteligencija bavi obradom prirodnog jezika je i komunikacija za čovekom. Iz tog aspekta, obrada prirodnog jezika uključuje dva dela. Prvi deo se odnosi na razumevanje i rasuđivanje nad ulaznim podacima koji su napisani prirodnom jezikom, a drugi na generisanje izlaza na prirodnom jeziku.

U ovom odeljku opisni su modeli jezika pogodni za korišćenje u komunikaciji. Modeli koji se koriste u komunikaciju moraju da omoguće dublje razumevanje teksta i zato su složeniji od modela koji se koriste za klasifikaciju teksta. U takve modele uključena su gramatička pravila, jezičke konstrukcije (frazе) i semantika jezika.

10.5.1 Formalni jezici

Na slici 10.2 prikazana je jedna rečenica na srpskom u formi stabla. Elemente *rečenica*, *imenski deo*, *glagolski deo*, *pridev*, *imenica*, *prilog* i *glagol* nazivamo neterminalnim simbolima, a listove stabla (*mala*, *deca*, *brzo*, *trče*) terminalnim ili završnim simbolima.



Slika 10.2

Formalnu gramatiku čine pravila oblika $\alpha \rightarrow \beta$ gde α i β mogu da sadrže terminalne i neterminalne simbole. Gramatika specificira i startni simbol koji je u obradi prirodnog jezika najčešće rečenica (eng. sentence). Na osnovu restrikcija na sadržaj α i β postoje sledeće vrste gramatika:

- kontekstno-osetljive gramatike kod kojih važi da desna strana pravila sadrži makar isto toliko terminalnih simbola kao i leva, na primer $ASB \rightarrow AAaBB$
- kontekstno slobodne gramatike sadrže pravila oblika neterminalni \rightarrow bilo šta, na primer $S \rightarrow aSb$
- regularne gramatike sadrže pravila oblika neterminalni \rightarrow terminalni[neterminalni], na primer $S \rightarrow aS$.

Popularni model jezika u problemima obrade prirodnog jezika je stohastička kontekstno-slobodna gramatika (eng. probabilistic/stochastic context-free grammar - PCFG/SCFG) koja podrazumeva specifikaciju verovatnoća u pravilima gramatike.

Primer pravila sa verovatnoćom je:

$VP \rightarrow \text{Verb}[0.70] \mid VP \text{ NP}[0.30]$.

Značenje verovatnoća u pravilu je da se glagolski deo (VP - verb phrase) sa verovatnoćom 0.7 sastoji od glagola, a sa verovatnoćom 0.30 od složenije strukture koja sadrži glagolski i imenski deo (NP - noun phrase).

Leksikon je lista reči dostupnih u jednom jeziku. Na slici 10.3 prikazan je leksikon za jezik sveta Wumpusa, koji predstavlja podskup engleskog jezika. Leksikon je prikazan u modelu jezika koji koristi verovatnoće.

<i>Noun</i>	\rightarrow	stench [0.05] breeze [0.10] wumpus [0.15] pits [0.05] ...
<i>Verb</i>	\rightarrow	is [0.10] feel [0.10] smells [0.10] stinks [0.05] ...
<i>Adjective</i>	\rightarrow	right [0.10] dead [0.05] smelly [0.02] breezy [0.02] ...
<i>Adverb</i>	\rightarrow	here [0.05] ahead [0.05] nearby [0.02] ...
<i>Pronoun</i>	\rightarrow	me [0.10] you [0.03] I [0.10] it [0.10] ...
<i>RelPro</i>	\rightarrow	that [0.40] which [0.15] who [0.20] whom [0.02] \vee ...
<i>Name</i>	\rightarrow	John [0.01] Mary [0.01] Boston [0.01] ...
<i>Article</i>	\rightarrow	the [0.40] a [0.30] an [0.10] every [0.05] ...
<i>Prep</i>	\rightarrow	to [0.20] in [0.10] on [0.05] near [0.10] ...
<i>Conj</i>	\rightarrow	and [0.50] or [0.10] but [0.20] yet [0.02] \vee ...
<i>Digit</i>	\rightarrow	0 [0.20] 1 [0.20] 2 [0.20] 3 [0.20] 4 [0.20] ...

Slika 10.3

Svaka kategorija u leksikonu jezika sveta Wumpusa (slika 10.4) završava se sa ... jer se u svakoj kategoriji nalazi još reči. Za neke kategorije, kao što su imenice, glagoli, prilozi je i nemoguće izlistati sve reči (u modelu prirodnog jezika). To su kategorije u kojima se neprestalno dodaju nove reči (iPod, biodisel) i takve kategorije se nazivaju otvorene. Postoje i zatvorene kategorije koje imaju mali broj reči (članovi-articles, predlozi, relativne zamenice) i koje se menjaju vrlo retko (tokom vekova).

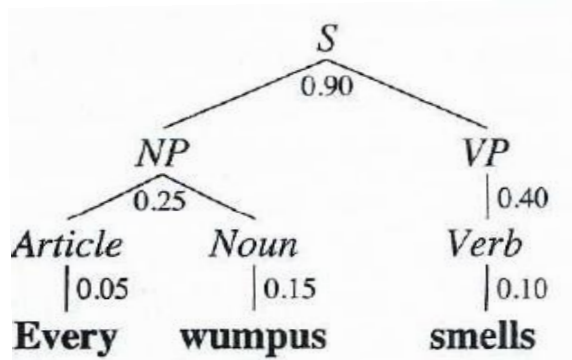
Na slici 10.4 prikazana je gramatika jezika sveta Wumpusa sa uključenim verovatnoćama. Neterminalni simbol S označava rečenicu (sentence) i predstavlja početni simbol. Simbol NP označava imenski deo rečenice (noun phrase), a VP glagolski deo rečenice (verb phrase). Pridevi su označeni simbolom Adj (adjectives), a konstrukcija koja sadrži predlog PP (propositional phrase). Pored svakog elementa sa desne strane pravila navedena je verovatnoća i primer jezičke konstrukcije tog dela pravila.

S	\rightarrow	$NP VP$	[0.90]	I + feel a breeze
		$S Conj S$	[0.10]	I feel a breeze + and + It stinks
NP	\rightarrow	$Pronoun$	[0.30]	I
		$Name$	[0.10]	John
		$Noun$	[0.10]	pits
		$Article Noun$	[0.25]	the + wumpus
		$Article Adjs Noun$	[0.05]	the + smelly dead + wumpus
		$Digit Digit$	[0.05]	3 4
		$NP PP$	[0.10]	the wumpus + in 1 3
		$NP RelClause$	[0.05]	the wumpus + that is smelly
VP	\rightarrow	$Verb$	[0.40]	stinks
		$VP NP$	[0.35]	feel + a breeze
		$VP Adjective$	[0.05]	smells + dead
		$VP PP$	[0.10]	is + in 1 3
		$VP Adverb$	[0.10]	go + ahead
$Adjs$	\rightarrow	$Adjective$	[0.80]	smelly
		$Adjective Adjs$	[0.20]	smelly + dead
PP	\rightarrow	$Prep NP$	[1.00]	to + the east
$RelClause$	\rightarrow	$RelPro VP$	[1.00]	that + is smelly

Slika 10.4

Na slici 10.5 prikazano je stablo parsiranja za rečenicu "Every wumpus smells". Verovatnoća ovog stabla parsiranja (i odgovarajuće rečenice) je proizvod svih verovatnoća koje su prikazane na stablu i iznosi $0.9 \cdot 0.25 \cdot 0.05 \cdot 0.15 \cdot 0.40 \cdot 0.10 = 0.0000675$. Posmatrana gramatika generiše veliki broj rečenica, neke

od njih imaju smisla kao što su John is in the pit, The wumpus that stinks is in 2 2, Mary is in Boston and the wumpus is near 3 2, a neke i nemaju, kao na primer Me go Boston i I smell pits wumpus John.

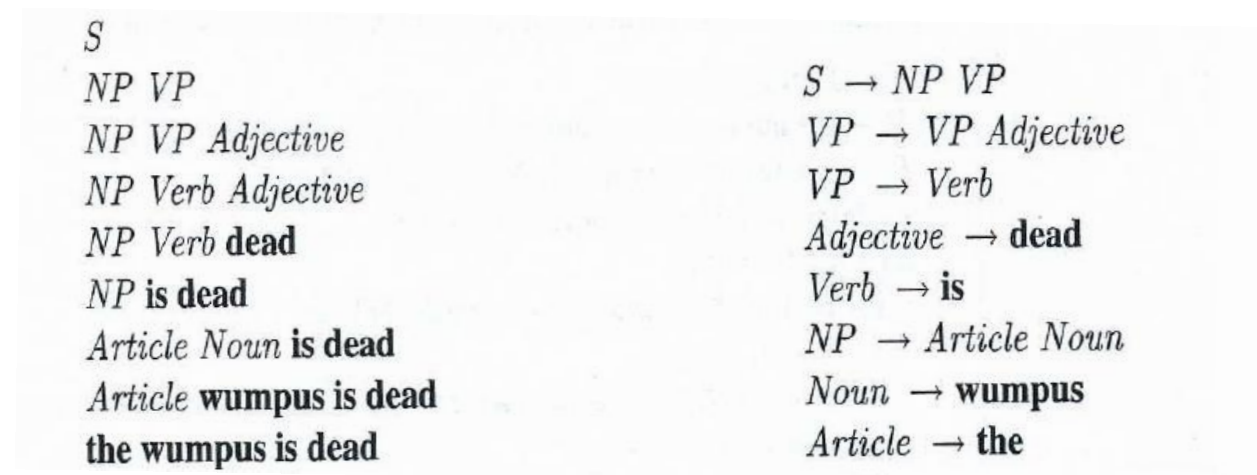


Slika 10.5

10.5.2 Sintaksna analiza

Sintaksna analiza teksta odnosi se na proces parsiranja kojim se tekst analizira i prepoznaje strukturu definisanu gramatikom. Na slici 10.6 prikazan je postupak parsiranja rečenice "the wumpus is dead".

Postoje dve vrste parsiranja to su top-down i button-up. Kod top-down parsiranja počinjemo od početnog neterminalnog simbola (S) i u svakom koraku vršimo uparivanje elemenata sa levom stranom pravila ($X \rightarrow Y$). Ako pronađemo ovakvo pravilo zamenjujemo X sa Y. Parsiranje buttom-up počinje od konkretne rečenice i u svakom koraku uparuje elemente sa desnom stranom pravila (Y) i ukoliko uparivanje uspe zamenjuje se Y sa X.



Slika 10.6

Parsiranja top-down i buttom-up su često neefikasna pri obradi prirodnog jezika. Na primer, ako posmatramo sledeće dve rečenice

Have the students in section 2 of Computer Science 101 take the exam.

Ove dve rečenice imaju isti početak do jedanaeste reči, a ipak su veoma različite i imaju različiti rezultat parsiranja, prva je naredba, a druga pitanje. Parser koji ide sa leva na desno treba da pogodi da li je prva reč deo naredbe ili pitanja, a neće znati da li je tačno pogodio sve dok ne dođe do jedanaeste reči. Ako nije pogodio, algoritam se bektrekom vraća na početak i analizira celu rečenicu sa drugom interpretacijom.

PCFG ima mnogo pravila, sa uključenim verovatnoćama i formiranje gramatike učenjem je prihvatljiviji pristup od inženjerstva znanja. Učenje kao obuku obično uzima korpus korektno parsiranog teksta koja se naziva treebank. Najpoznatiji je Penn TreeBank koji sadrži oko 3 miliona reči sa anotacijama za delove teksta i stablo parsiranja, kreirali su je ljudi ručnim unosom uz korišćenje nekih softverskih alata. Učenje verovatnoće se vrši jednostavnim prebrojavanjem.

Gramatike se mogu proširiti kako bi se neke konstrukcije vezale za kontekst.

Da bi se uspostavio odnos između reči „eat“ i reči „banana“ nasuprot reči „bandanna“, koristi se leksikalizovani PCFG (lexicalized) u kom verovatnoća u pravilima zavisi od odnosa reči u stablu parsiranja, a ne samo od položaja reči u rečenici. Ovde se uvodi pojam glava fraze koje predstavlja najznačajniju reč u frazi, na primer, u VP konstrukciji „eat a banana“ glagol „eat“ je glava, a u NP konstrukciji „a banana“ glava je reč „banana“. Koristimo notaciju VP(v) za označavanje fraze koja je oblika VP, a čija je glava v. Na ovaj način pravimo proširenje gramatike odnosom glagol-objekat koja sada izgleda ovako:

$$\text{NP}(n) \rightarrow \text{Article}(a) \text{ Adj}(j) \text{ Noun}(n) \quad [\text{P}_3(j,n)]$$

U ovakvoj gramatici verovatnoća $P_1(v,n)$ zavisi od glava v i n . Ova verovatnoća će biti veća kada je v „eat“, a n „banana“, a manja ako je n „bandanna“. Pošto posmatramo samo glave, u ove verovatnoće biće uključena i konstrukcija „eat a rancid banana“.

Gramatika i logika

Ovakva proširena gramatika se može opisati i formalno u logici prvog reda korišćenjem definite clauzula (definite clauses) i naziva se definite clause grammar ili DCG. Definite klauzule su disjunkcije literala u kojima je tačno jedan pozitivan (na primer. $\neg A \vee \neg B \vee C$).

Posmatrajmo sledeće produkciono pravilo gramatike:

$NP(n) \rightarrow Article(a) Adj(j) Noun(n) \{Compatible(j,n)\}$

Novi koncept je {constraint} koji označava ograničenje nad promenljivima i pravilo važi samo ako je zadovoljeno ograničenje. Predikat $Compatible(j,n)$ označava da su pridev j i imenica n kompatibilni i definiše se skupom iskaza, na primer $Compatible(black, dog)$. Ovakvo pravilo možemo da predstavimo kao definite clauzulu na sledeći način

$Article(a, s_1) \wedge Adjs(j, s_2) \wedge Noun(n, s_3) \wedge Compatible(j, n) \Rightarrow NP(n, Append(s_1, s_2, s_3))$.

Ovakva logička rečenica znači da ako je predikat *Article* tačan za glavu a i string s_1 i *Adj* za glavu j i string s_2 i *Noun* za glavu n i string s_3 i ako su j i n kompatibilni, tada je predikat *NP* tačan za glavu n i string koji se dobije konkatenacijom stringova s_1, s_2 i s_3 . Ovde su izostavljene verovatnoće, ali se one mogu jednostavno dodati kao argumenti predikata sa leve strane implikacije, a verovatnoća na desnoj strani implikacije je jednaka proizvodu tih verovatnoća.

Predstavljanje pravila gramatike u logici prvog reda nam omogućava da parsiranje posmatramo kao logičko rezonovanje (logical inference) i da obradu jezika vršimo pomoću pravila zaključivanja u logici. Na primer, parsiranje bottom-up se može zameniti za ulančavanjem unapred, a top-down sa ulančavanjem unazad. Zbog ovoga je obrada prirodnog jezika sa DCG-om jedan od prvih primena logičkog programiranja u Prologu i motivacija za kreiranje ovog jezika. Pored parsiranja, logičko rezonovanje se može koristiti i za generisanje jezika.

10.5.4 Glavni problemi obrade prirodnog jezika

Gramatika prirodnog jezika je veoma kompleksna i njena kompletna specifikacija ima brojne komplikacije, kao što interpretacija vremena, kvantifikacije i dvosmislenost. Postoji više vrsta dvosmislenosti, jedna je leksička dvosmislenost koja znači da reč može imati više značenja (na primer back, može biti prilog - go back, pridev - back door, glagol - back up your files). Druga dvosmislenost je sintaksna dvosmislenost koja se odnosi na to da fraza može da se parsira na više načina, na primer rečenica "I smelled a wumpus in 2,2" ima dva moguća parsiranja, jedan u kom se fraza "in 2,2" odnosi na glagol, a jedna u kom se odnosi na imenicu. Sintaksna dvosmislenost vodi do semantičke dvosmislenosti jer ista rečenica kada se isparsira na dva opisana načina ima potpuno drugačiju semantičku interpretaciju.